LATENT SEMANTIC WEB SERVICE DISCOVERY AND COMPOSITION FRAMEWORK

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

(Winnie) Yuki B. Yick

Aug 2009

COMMITTEE MEMBERSHIP

TITLE:	Latent Semantic Web Service Discovery and Composition Framework
AUTHOR:	(Winnie) Yuki B. Yick
DATE SUBMITTED:	Dec. 2009
COMMITTEE CHAIR:	Dr. Michael Haungs
COMMITTEE MEMBER:	Dr. Phillip Nico
COMMITTEE MEMBER:	Dr. Aaron Keen

ABSTRACT

Latent Semantic Web Service Discovery and Composition Framework

(Winnie) Yuki B. Yick

A web service is a software system that supports interoperable machine to machine interaction over the network. It can be any kind of service provided on the web that can exchange data between applications. In simple terms, it is any program that is callable by another program across the web using standard protocols. Web services are special because they are independent of the platform, programming language, and model design. In current web applications, more businesses are gradually publishing their business as services over the web. This growing number of web services available within an organization and on the Web raises a new and challenging search problem: locating desired web services. Searching for web services with conventional web search engines is insufficient in this context. In this paper, a latent semantic web service discovery and composition framework, addressing both discovery and composition of web services, is introduced. The framework provides a graphical visualization interface that can facilitate users in finding the desired web service beyond the general keyword search and provide composition of complex services if the service does not exist. A novel approach based on an information retrieval technique known as latent semantic analysis is applied to a large web service descriptions data set collection. By using co-occurrence patterns across the entire data collection, and then using those patterns to infer semantic relationships between documents, relevant results in a ranked order can be retrieved. Moreover, a back to front algorithm is applied to the composition search interface in order to compose complex web services in full or partial terms from pre-existing web services when the requested service does not exist. An experimental study conducted on a collection of 2525 publicity available web service shows improved performance to the technique applied in obtaining much better search results.

ACKNOWLEDGMENTS

First and foremost, I am thankful and offer my sincerest gratitude to my advisor, Dr.Michael Haungs, whose encouragement, guidance and support from the initial to the end enabled me to develop an understanding of the subject.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of my thesis.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table

Page

LIST OF EQUATIONS

Equation
Page

LIST OF FIGURES

Figure Page

1. Introduction

Over the past decades the Internet has gone through a staggering growth. Some sources estimate that over twenty percent of global population already has access to the Web [1]. The Web currently has over 4 billion pages and about 1 million added every day [2]. Such growth is in part due to the constant advancements in the computing technology. Rapid advancements in computer and networking technology are making access to the Web and the computing devices that connect to it cheaper than ever before, allowing more and more people to connect to the Internet. In addition to decreasing the cost of access to the Internet, advancements in technology, such as access through mobile phones and affordable high speed connections, are responsible for the ever increasing complexity of how we use the Web. This ability to use the Web in diverse ways is one of the reasons that attracts so many users to it and drives the technology that improves it. Whereas HTML was sufficient for most tasks just a few years ago, now there are a plethora of technologies that have capabilities far exceeding that of HTML that fulfill the users need for diverse services. There are services offering streaming video, dynamic content sites, online games, online retail stores, online digital music distributions, banking, etc. This rise in variety of use drives the need for increasing complexity of web applications. Specifically, one of the pertaining problems with web applications lies in finding the needed information in the sea of growing data that is currently in the Web. As such, the search engine became an essential part of the World Wide Web providing the directory of Internet content indexed and searchable for users to locate desired

information on the Web. A related problem to the web search faces service providing entities such as businesses and government agencies. The recent trends show that the Web is moving towards application to application communication [3], enabling business to do business to business communication faster than before. The ability for applications to seamlessly exchange data amongst themselves is becoming more practical and advantageous [4]. The problem is twofold; how to have the highest possible service visibility to the targeted users, and, how to offer a standard interface to the user in locating the desired web service so as to allow minimal software development. With more web services on the rise, we present how web services can be useful.

1.1 Uses of Web Service in Applications:

The main idea behind web services is that software can now be a service. It allows business to model their business functions or operations as in and out services in order to integrate with other businesses for faster business communications. Aside from the business point of view, on the software perspective, web services allow for the reuse of code on the web, and easier system integration. Software developers can spend less time reproducing an already existing method, integrating with legacy systems, and or providing service that is already publically available and focus on the main business functional tasks. A few detailed examples of different aspects when using web services follows:

1.1.1 Communicating through a Firewall

When an application needs to communicate between the client and the server and is normally conflicted with problems of the firewall or proxy servers, exposing the application components as web services would be one beneficial solution. One example of such a case is a distributed application with many users spread over different locations and requires intensive interaction between the user interface and middle tier. In this case, a browser-based client and active server pages (ASPs) are typically used to expose the middle tier business logics to the user interface. Such an architecture is difficult to develop and hard to maintain. In order to create a new screen for the application, a user interface design of a web page and the corresponding middle tier components of business logic behind the new screen would need to be created. In addition, one or more active server pages is required to receive input from the user interface which invokes the middle tier components, formats the results as HTML, and sends results to the browser.

A web service can levitate this extra step of building the ASP here and create a client less dependent on HTML forms. By exposing the middle tier business logic components as web services and invoking them directly from a user interface running on client machines, a cleaner and reduce development time, code complexity and maintainability can be achieved. Moreover, by using web services to expose the application logic and data provides a fundamental

foundation for reusing your layer of web services for any other purpose or for the application from any client running on any type of platform [11].

1.1.2 Application Integration

Integration is another benefit of using web services. Since many businesses applications are continually being written in various languages and running on disparate systems, there is a good portion of time spent in integrating applications. Many businesses still run data to and from legacy applications running on a mainframe or using applications from different vendors where data often needs to be integrated. Data integration can be made easier if applications can expose functionality and data as a web service providing a standard mechanism for other applications to integrate with it. An example usage in this case is when different vendor applications are used. A retail company might have an order-entry application that is used to enter in new orders, customer information, shipping addresses, quantities, prices, and payment information. An order shipment application from another vendor in the warehouse might be used to take an order product and ship it as seen in Figure 1. When a new order is placed, the order entry application would notify the shipment application to ship the order. Using a web service layer on top of the shipment application, an "AddOrder" function can be exposed as a web service for example that would allow the order-entry application to invoke the function each time a new order is placed [11].

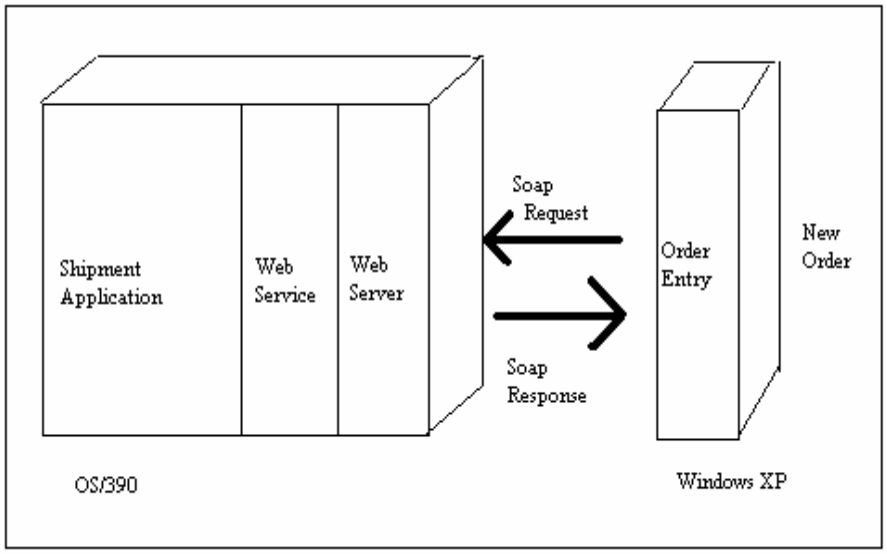

Figure 1: Integrating applications with Web services

1.1.3 Business-to-Business Integration

A Business-to-Business (B2B) Integration is needed when business processes span across multiple businesses either as to suppliers or customers. With web services, business integration can be easily done by exposing those certain business processes to authorized suppliers or customers. Web services are not a new concept of doing things or change the way tasks are completed, but it simply makes implementation and maintenance easier. For example, a business retailer may use a concept called Electronic Document Interchange (EDI) to process customer purchase orders and supplies to send invoices. The National Institute of Standards and Technology in a 1996 publication defines Electronic Data Interchange as "the computer-to-computer interchange of strictly formatted messages that represent documents other than monetary

instruments. EDI implies a sequence of messages between two parties, either of whom may serve as originator or recipient. The formatted data representing the documents may be transmitted from originator to recipient via telecommunications or physically transported on electronic storage media." [14]. In 1992, a survey of Canadian businesses found at least 140 that had adopted some form of EDI, but that many (in the sample) "[had] not benefited from implementing EDI, and that they [had] in fact been disadvantaged by it" [12]. The key difference here is that, web services can operate over the Internet for fast delivery and are much easier to implement than the EDI method [11].

1.1.4 Software Reuse

Additionally, software reuse is the most promising. By exposing an application's functionality as web services, code porting and conversion is not necessary. Applications can now be connected to any other applications from legacy systems or different programming languages. Reusing static data where it used to be a problem to store is now possible and can be reused across different applications. Single portals can now be created to combine many applications as a single service. For example, a FedEx shipping application, stock quote verifier, view a calendar, purchase tickets can all be combined in a single portal.

1.1.6 When web service is not necessary to do the job

Situations where using web services may not be of great benefit are using a single machine application and homogeneous applications on a LAN. When applications that are not to be shared or stand alone would not require web services.

The focus of this paper is to describe a frame work that facilitates the discovery of web services through a technique of Latent Semantic Analysis and provide web service composition. The frame work is composed of an organized repository of a set of public web services and exposes a friendly graphical interface for obtaining this information. Dedicated web crawlers are designed to automatically find and retrieve existing public web services on the web into the repository. In addition, the framework organizes the retrieved web service using Latent Semantic Analysis (LSA) to understand and organize the data for better search results. The graphical interface for users present results in an efficient human readable form. The presentation of the results is ranked by the most related search results. Retrieval of the service provider's location, web service description in full detail and the input output parameters displayed in a tree structure is presented. Additionally, a graphical view of latent semantic web services is presented as a node connected graph. A composition search interface is also provided to create new web services. The use of the graphical search interface is intended to speed up and reduce the search time for users to find the right web service.

The rest of this paper is organized as follows. First, a review of the background material is discussed. Next, related work is given. The third section presents the design and implementation of the overall web service framework. Experiments and data analysis of experimental results are presented in the fourth section. The last sections talk about future work and conclude the paper.

2. Background

With more web services becoming readily available, searching for the right service for your application through the World Wide Web can become difficult. The originally proposed solution to handle this problem was the Universal Description Discovery and Integration (UDDI). With UDDI, a business can: publish their web service listings, discover web services in the registry, and provide information on how a service or software application can interact over the Internet [5]. UDDI provides an API for developers to query the registry by a keyword search on service names or by searching within a category. Information about the web service and the service provider is returned. The user can then communicate with the service by the provided Web Service Description Language (WSDL) document that is published by the service provider at a URL location. The WSDL document is a XML based document that describes all the necessities of the web service. Specifically, it contains the web service interface invocation method for that service and the location of the service provider. The search request to UDDI is done by sending messages to the registry. The exchange of messages is usually done through SOAP, which is a simple XML based protocol that allows applications to exchange data over HTTP. With over 50,000 services registered

since 2000 in its directory and as more web services continue to be added, methods of finding the right service by a keyword becomes challenging. At the basic level, UDDI's API provides only a *simple keyword search* on the web service name. Keyword searches, in general, are insufficient because they do not capture the underlying semantics of the web service. For example, when searching for *automobile*, the web services whose descriptions contain the term *car*, *truck*, *auto* or *transportation* but not *automobile* will not be returned.

In January of 2006, the public UDDI registry hosted by Microsoft, IBM and SAP was officially shutdown. No new services were allowed to be published into UDDI. Since the intended project to prove the interoperability and robustness of the UDDI registry was met and provided the proof of concept for the early stages of web services business applications [6], the disappearance of a dedicated web services registry has made service discovery an increasingly difficult task. To find a desired service, a user has to either contact the potential service provider directly (assuming the user knows who that is) or turn to the conventional web based search engines. Most businesses have their own internal business UDDI, but the use of many already existing useful public services are at loss. The only drawback to the conventional web based search engines is that the web services found depends on the portion of the Internet that is crawled by the search engine. Additional, the search engine indexes WSDL documents like that of a regular HTML web page. This method lacks the full potential of finding a web service because of the difference between a WSDL document structure and a web page structure in capturing information for its index. When a user is looking for a specific web service,

many results would be returned from a regular web based search engine and the user would need to filter both web data content from a web service [7]. Thus, the need for a specific web service discovery engine is necessary to provide users with only web service based searches and beyond the general *simple keyword search*.

UDDI was a proof of concept created in the early stages of web services development. As such, it was designed with simple features in mind, limiting its usability. Specifically, UDDI was a registry that was meant to be accessed via its API directly by submitting a single message query to the registry server. As such there was no adequate human interface. Since the registry contained over 50,000 services, it became often the case that a user would want to manually search the database for the available services. By querying UDDI directly, a user would get a list of WDSL files, which are scripted files difficult to comprehend.

A related problem occurs when a desired web service does not exist, but group of related web services exists that can be composed to derive the required functionality. This is a topic of web service composition, an active research area in the industry and academia. It thus becomes an important feature of a web service search engine to provide a possible composition of existing web services that fulfills the query if the desired web service is not found. This provides web service reusability and removes the need to create redundant web services that can be done through already existing services.

The focus in discussion is mainly on XML web services for interfacing application to application communication based on XML message exchange. A web

service is generally used with standards such as SOAP(for message exchange), WSDL(web service description language), and UDDI(universal description discovery Integration). Although these standards are used when describing a web service, it is not mandatory requirements of a web service. The service interface is generally described by a WSDL document which is a machine readable XML format. Systems can interact with the Web service following the WSDL service invocation description and using as transport SOAP, HTTP get/post, or MIME [3]. A web service is normally used to provide some functionality on behalf of a business or individual. Basically, in simple terms, a web service is a collection of functions that a developer packages as a single entity and publishes on a network for use by other programs [8]. The benefits of a web service are that it provides independence and service reusability across different frameworks and different platforms. Web services form the building blocks to constructing web applications more efficiently [9]. It has been the cost effective solution for uniting information that is distributed between applications over different platforms [5]. One of the benefits of using a web service as to other methods is the cost and ease of building the service since it uses already existing web protocols for data transfer.

The general framework of a web service can be described as follows for understanding the fundamental workings of web services. A service provider pertaining to a business would provide a service and the user who is the service requester would use the provided service. The service provider would provide its services by publishing to the UDDI registry. The WSDL document is used to describe the service as a set of ports and necessary binding governing the interaction between the service provider and the

service requester. These ports are the operations that are allowed for data exchange given input messages and optional output message [9]. Soap is normally the binding chosen as the transport for distributed applications to transfer data in a XML based format over HTTP. Other bindings that may be used include HTTP Get/Post requests and MIME. The client/user would discover a service by sending a query request to the UDDI registry. When the client finds the needed service, it would retrieve the service provider's information in order to contact the service provider for the service information. The user requests for usage of the service by sending a SOAP message request and waiting for the appropriate reply [10]. Figure 2 below shows a diagram of the message exchanges between a service being published, discovered, and invoked.

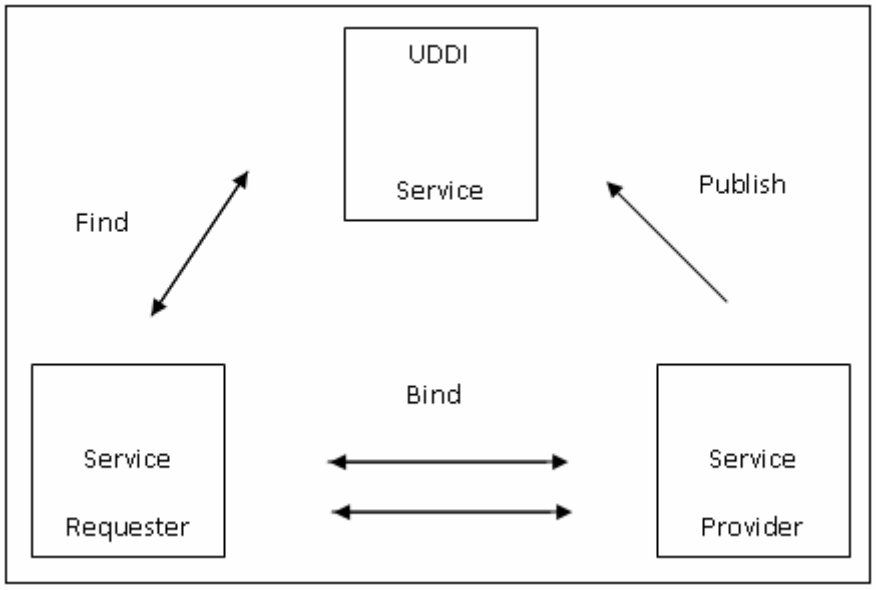

Figure 2: Diagram of web service process

The WSDL document is written in XML and provides a way for service providers to describe the basic formats of service request of web protocols. The main structure of a WSDL specification defines the following elements types, message, operation, port type, binding, port and service.

- <Types> -provides the data type definitions used to describe the messages parameters being exchanged.
- <Message> -represents the data being transmitted or communicated.
- <Operation> –is the operation provided by the service. Similar to functions in a class, it refers to an input message and output message.
- <Port Type> -is a set of operations supported by one or more endpoints.
- <Binding> -specifies the protocol and data format specification for the operations and messages defined by a particular port type.
- <Port> -specifies an address for a binding or the URL where the web service is listening.
- <Service> -is a collection of related ports.

Below in Figure 3 is a partial sample WSDL file of the weather service named WeatherForcast located at http://s1.devel.sip.su.se/WeatherForecast.yaws. The partial file shows an example of the web service named WeatherForecast. It has three types of binding available: SOAP, HTTPGet, and HTTPPost. One of the available port type's name is WeatherForecastHTTPGet. It contains two operations: GetWeatherByZipCode and GetWeatherByPlaceName. One of the input messages of the operation is GetWeatherByZipCodeHTTPGetIn containing the message input parameter ZipCode that is of string data type.

```
<wsdl:message name="GetWeatherByZipCodeHttpGetIn">
<wsdl:part name="ZipCode" type="s:string"/>
</wsdl:message>

<wsdl:portType name="WeatherForecastHttpGet">
–
<wsdl:operation name="GetWeatherByZipCode">
–
<documentation>
Get one week weather forecast for a valid Zip Code(USA)
</documentation>
<wsdl:input message="tns:GetWeatherByZipCodeHttpGetIn"/>
<wsdl:output message="tns:GetWeatherByZipCodeHttpGetOut"/>
</wsdl:operation>
–
<wsdl:operation name="GetWeatherByPlaceName">
–
<documentation>
Get one week  weather forecast for a place name(USA)
</documentation>
<wsdl:input message="tns:GetWeatherByPlaceNameHttpGetIn"/>
<wsdl:output message="tns:GetWeatherByPlaceNameHttpGetOut"/>
</wsdl:operation>
</wsdl:portType>


<wsdl:service name="WeatherForecast">
–
<documentation>
Get one week weather forecast for valid zip code or Place name in USA
</documentation>
–
<wsdl:port name="WeatherForecastSoap" binding="tns:WeatherForecastSoap">
<wsdlsoap:address location="http://s1.devel.sip.su.se/WeatherForecast.yaws"/>
</wsdl:port>
–
<wsdl:port name="WeatherForecastHttpGet" binding="tns:WeatherForecastHttpGet">
<http:address location="http://s1.devel.sip.su.se/WeatherForecast.yaws"/>
</wsdl:port>
–
<wsdl:port name="WeatherForecastHttpPost" binding="tns:WeatherForecastHttpPost">
<http:address location="http://s1.devel.sip.su.se/WeatherForecast.yaws"/>
</wsdl:port>
</wsdl:service>
```

Figure 3: Example of partial WSDL file: Weather.wsdl

3. Latent Semantic Web Service Discovery and Composition Framework

In order to provide a more suitable search for web services to clients' needs, a specific web service search engine is needed to deliver only prevalent web service results. The latent semantic web service framework would provide an extensible keyword search allowing similar relevant candidate services to be found. Sometimes a user may not know or have the right keyword in finding the right service for their application, a good user interface (UI) can guide the way to better search results. If a users keyword does not match an exact web service name, it would be desirable to formulate a query that can deliver all similar services that may be related to the users intended service. The query composer is responsible for analyzing the input and to formulate additional word association using WordNet (a lexical dictionary developed at Princeton University [34]) into the query search for all related services. This can broaden or find semantically related services with the extended keywords.

A specific purpose WSDL crawler that works as the backend of the web service discovery composition engine would search the Web for all public web services. Based on an investigation done by [7], the best method for finding the most active web services is via the web, specifically search engines that constitutes 72% of active web services. The only downside to searching for web services via web based search engines is that it lacks the potential for a user to find their specific service needs from the large search

result set. As general web crawlers only fetch the assessable WSDL documents from the server site, it has no content information about the cached or stored WSDL document information. The WSDL document is then parsed like regular web page contents in creating the index for the general web search engines. As a result, web based search engines do not provide a well generated index for finding web services. Using this knowledge, our WSDL crawlers are specific to discover solely web services by crawling the web for all web service description documents and then extracting content information and creating a web service index based on the collected service repository for the Latent Semantic Web Service Discovery Composition Framework. Having a specific web service discovery composition search engine would allow the service provider's web service to be found and explored to its full potential.

Upon finding a web service, the service will be validated and the WSDL document will be retrieved into a central repository. All the WSDL for each service found would be stored into a repository and indexed. The backbone of the discovery composition framework is done using a novel data mining technique of latent semantic analysis (LSA) to extract knowledge from the WSDL documents. Using latent semantic analysis, similarities can be found among the WSDL web services in order to gain improvements in search result precision, recall and ranking. The way latent semantic analysis works is that it looks at patterns of word distribution (specifically, word co-occurrence) across a set of documents [13]. By analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the

documents and terms, search queries will be able to retrieve documents and all concept related documents as well that are similar.

In many information retrieval research, the techniques that apply mostly deal with HTML web content. Since WSDL documents are not similar to that of web content with only available short service names, parameter names, etc that are mostly short phrases or incomplete sentences, it does not contain the necessary information in web pages that can infer knowledge or information needed for content retrieval indexes. Due to the unknowing nature of the structured text, applying information retrieval techniques may or may not be effective. Therefore, LSA is chosen as the method to test since it does not need to be in sentence form for LSA to be effective. It can work with lists, free-form notes, email, Web-based content, etc., as long as a collection of text contains multiple terms, LSA can be used to identify patterns in the relationships between the important terms and concepts contained in the text.

The first step to LSA is finding content words with semantic meaning in documents and using those words to find patterns within the document set. Having the list of content words, a term-document matrix can be generated. The term-document matrix is basically a large grid containing information about all documents in the set and what terms pertain to which documents. This large grid can be useful in search lookups for finding words in documents. Due to the large grid's size to the size of the document set, a reduction by singular value decomposition (SVD) is used to decompose the matrix. The mathematics behind singular value decomposition is that it decomposes a matrix in high dimensions to a much smaller dimensional space, much like an optimal mapping

between points in 3D space to a 2D space or plane. During the reduction, the algorithm tries to get the best optimal data loss, and content words get superimpose on one another. This mapping in SVD is what also provides data concepts to arise when the words merge down into similar groupings. Once SVD is complete, search retrieval can be improved by finding similar documents returned by a SVD search query. Additionally, WordNet is used to create similar lists with words that have the same meaning as part of the search query for broadening the web search and increasing recall. Another benefit of using LSA is that results can be ranked easily. Each web service will also be presented in a graph view providing connections to its closest service related neighbor. The full WSDL document and a quick view of input/output parameters will also be presented for users' convenience and search needs.

If a service does not exist in the repository, the composition engine would then compose existing services to fulfill the users' requirements. In order to provide service composition, it will look for functionalities of existing service components that can be integrated to accomplish the requested service. The service composition algorithm will be based on a binary tree method from BITS [15]. The discovery composition engine differs from BITS in that it will extend its result set by providing a partial service composition when no compositions are found. Compared to BITS, the algorithm will only look at the Cartesian product results of the minimum web services needed to create the composition instead of all possible results. It will also use real world web services for testing the nature of web service composition behavior. Returning a partial result when a full composition is not possible will be helpful to the user in that the user can

build from the partial services to create the missing component needed instead of writing the whole service from scratch.

4. Related work

Research in web services has been done in both areas of service discovery and service composition. Most work done in service discovery include semantic or syntactic matching of keyword by either a mixed or combination of exact and or part name matching of a web service's service name, operation name, parameter type, or input output parameter names [21, 22, 23, 24, 25, 26]. These searches are done with a single keyword match. With different matching parameters and criteria, each paper uses a different index for retrieving services. As in [22], Service names, along with operation names are used to create relationships to group services in the same is-a relationship or instance of relationship category to create a service index and a partname index for keyword matches for more efficient retrieval. Research of [21] uses a syntactical matching method of using part names of input and output messages. Three indexes – input index, output index, and operational name index are used for finding the right service and do speed up the discovery process. There are also researches in areas of signature matching, structure matching and specification matching for component discovery [27, 28, 30, 31]. Others have tried using DAML and DAML-S which is a formal logic based language that supports the specification of semantic information in RDF format. But currently due to its high cost and low widespread adoption, the move toward DAML is unlikely [29].

Research in web service composition has mostly been based on restricted input and output parameters to comply with the client's needs. Most work uses a variation of the algorithm of back to front searching [21, 22, 23, 15]. The algorithm starts from the output parameter match and searches up the chain for inputs until the desired output and input parameters match the requested service request. If a match is found then the service is satisfied with the combination of services at hand. This method was chosen as the better algorithm as it avoids many repeated tree node visits from the front to back algorithm. Another alternative is also seen in [33, 32] a different approach to composition is taken. One approach is the web services are orchestrated with no restriction on input and output parameters but into finding the separate services that can satisfy the users request given that there are no preset input/output parameters restrictions. An example can be of finding an electronic library given you combine a library search service with a payment service.

While most research separates discovery of web services and web service composition as separate areas of study. My paper differs from these methods in that it provides a combined study of service discovery and composition in providing a useful service discovery and composition GUI framework.

5. Design & Implementation

In this section, the architecture design of the overall framework is described in detail. The prototype implemented follows is described in full here for each of component's that makes up the framework.

5.1 Design of overall framework requirements

The crux of this research paper lies in the continually growing mass availability of the web services on the Web. Specifically, as of shutdown, UDDI registry indexed over 50,000 services. Since web services are now dispersed around the Web, their discovery has become a daunting task. This is true especially for the recently published services. Without central system, such as UDDI, dedicated to finding and registration of web services, their discovery is done primarily through conventional web searching services provided by entities like Google and Yahoo. The problem here lies in that these web search engines are general in nature, as they try to index as much of the Web as possible. Web services are specific type of information. Hence web service location queries to these search engines have to be carefully structured. This process is complicated by several factors.

First, none of the major search engines allow filtering of XML files. Most web services are described in XML format. Hence, a search query to a general search engine ideally filters non XML documents in the results. Second, the notion of a web services is fluid. This raises a problem for discovery since there is no strict format to which a web services must adhere. Although many services descriptions are written in WSDL format, there are competing formats, such as OWL-S, Microsoft .Net services, or a mere description on a web page. Hence, more complex filtering of query results is required. Thirdly, a major part of this research consists of the web service composition. Composition of two or more services may be possible in situations when a desired service is not available, but other services are available such that their composition provides the

desired functionality. However, if a desired service is not found via traditional web search, it is not clear what candidates for composite service to search for. Thus, this paper addresses the need for an interface that can discover web services and composite those services that are not available at hand.

The diagram of Figure 4 below shows the overall framework organizational structure. As can be seen in the figure, users interact with the framework by a graphical user interface. The interface accepts a user action and sends a request to the server for data. The server is build upon a web service repository that contains all the public web services retrieved by specific web service crawlers. The WSDL documents are processed into indexes and the query composer formulates the query and delegates the task for searching or composing of the services.

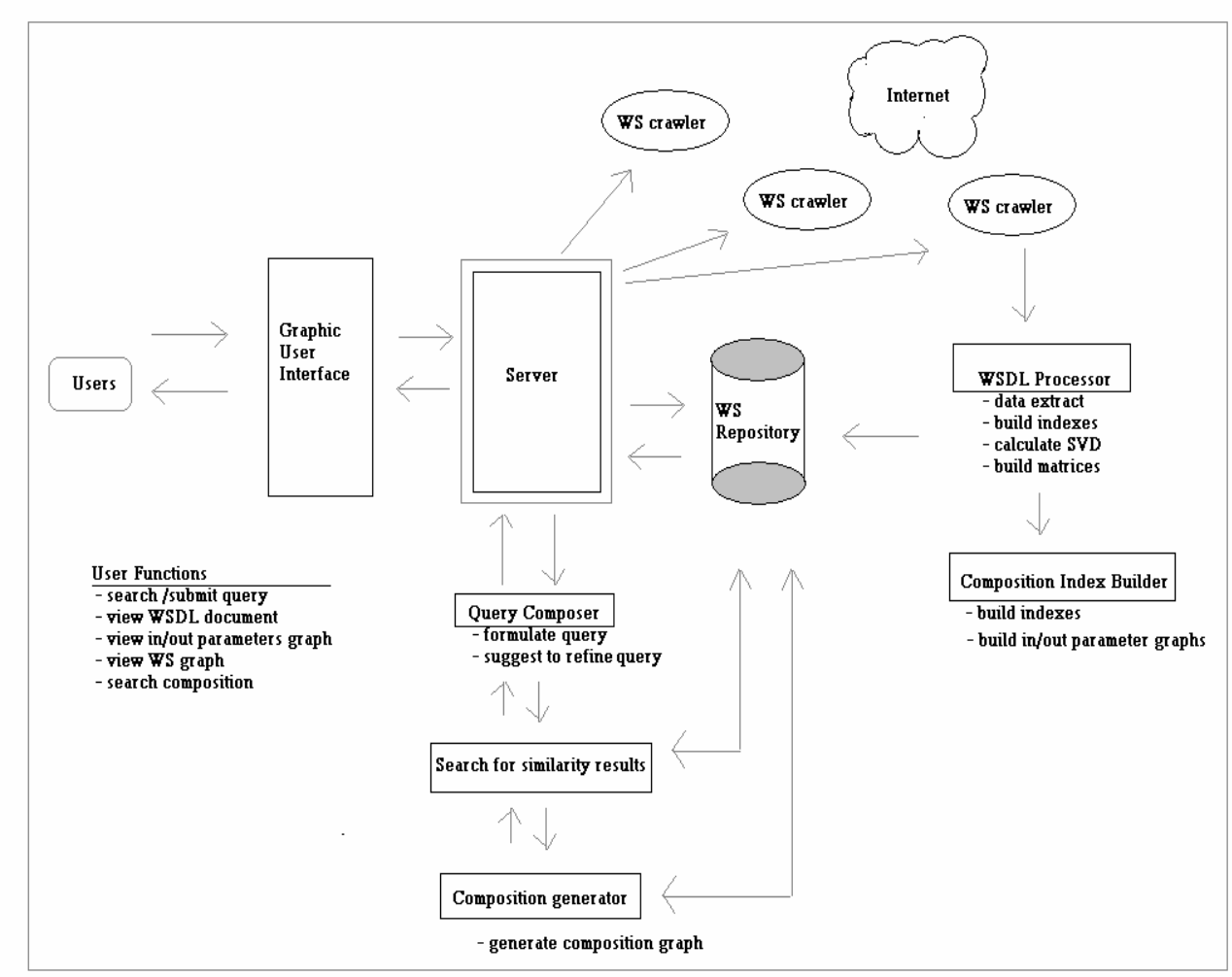

Figure 4: Latent Semantic Discovery and Composition Framework Organization

This paper addresses the issue of service location in two steps. The first step involves acquiring data from the web. This is done through a traditional approach of web crawling discussed in the next section of 4.2.1. The second step involves filtering the acquired files for valid web services and extracting data in 4.2.2. Section 4.2.3 discusses how data is analyzed through LSA to build the necessary matrices and indexes for the search repository. The composition algorithm will then be discussed in Section 4.2.4.

Section 4.2.5 will discuss the query composer and section 4.2.6 will specify the graphical user interface and how a user can utilize the GUI for an efficient web service discovery process in selecting the most appropriate web service from the list of potential services.

5.2 Implementation details

Each component of the framework will be discussed in the sections to follow. A detailed implementation explains the main function of how each component contribute to the overall framework. Further details can be viewed in the programming code.

5.2.1 Building the Specialized Web Service crawler:

Since the goal of the needed web crawler is to specifically find web services, the type of crawler used is categorized as a focus crawler. The details of the crawler are given below. Due to the ever growing size of the internet, and importance of web search, there are many different advanced crawling techniques. A study by Lawrence and Giles showed that no search engine indexes more than 16% of the Web and it is desirable that the downloaded fraction of the Web be as relevant pages rather than a random sample of the Web [16]. In order to get a good fraction of the Web, good seeds are needed for starting points. Since crawling in itself is not the focus of this paper, nor do I have the computing resources to crawl a reasonable percentage of the web, the architecture of the designed crawler is not written for optimal efficiency. Instead the crawler is provided as a point of reference and a foundation for the rest of the project. Specifically, during a test, the crawler ran for approximately nine hours, and found 30 web services. Since

obtained results are not an adequate proportion of available data, the crawler was modified to delegate the crawling to standard search engines such as Google and other search engines for seeding the WSDL crawler and as the crawling frontier. The following seeds were used during the crawl of the dataset: google, yahoo, msn, salcentral, xmethods, and seekda. The WS crawler is implemented in PHP script. The descriptions of the modification, as well as results of the crawler, are described below.

The following section will first introduce the focused WSDL crawler of the discovery composition engine, following a modified version using general search engines as the starting seed crawler frontier. The WSDL crawler first starts with a list of URLs called the seeds. The selection of seeds is obtained through the open directory project dmoz.org and search engines. Then, the URLs are fetched, see Figure 5. The pages are crawled recursively by saving pertinent data and extracting links. Using regular expression grammar, a match is conducted on the page content for extraction of certain important data. An example of regular expression matching on hyperlink extraction is shown here in Figure 6.

```
function get_page( $url )
{
        $options = array(
                CURLOPT_RETURNTRANSFER => true,     // return web page
                CURLOPT_HEADER         => false,    // don't return headers
                CURLOPT_FOLLOWLOCATION => true,     // follow redirects
                CURLOPT_ENCODING       => "",       // handle all encodings
                CURLOPT_USERAGENT      => "WScrawler", // who am i
                CURLOPT_AUTOREFERER    => true,     // set referer on redirect
                CURLOPT_CONNECTTIMEOUT => 120,      // timeout on connect
                CURLOPT_TIMEOUT        => 120,      // timeout on response
                CURLOPT_MAXREDIRS      => 10,       // stop after 10 redirects

        );

        $ch      = curl_init($url);

        curl_setopt_array( $ch, $options );
        $content = curl_exec( $ch );
        $err     = curl_errno( $ch );
        $errmsg  = curl_error( $ch );
        $header  = curl_getinfo( $ch );
        curl_close( $ch );

        $header['errno']   = $err;
        $header['errmsg']  = $errmsg;
        $header['content'] = $content;

        return $header;

}
```

Figure 5: Function to retrieve web page contents

```
$pattern='/<a\s+.*?href=[\"\']?([^\"\'>]*)[\"\']?[^>]*>(.*?)<\/a>/i';
if( preg_match_all ( $pattern, $page_content, $results, PREG_SET_ORDER ))
{
        foreach( $results as $match )
                array_push($linkArray, array( $match[1], $match[2] ));
}
```

Figure 6: Code snippet to extract all hyperlinks in a retrieved web page

The extracted links are then placed in a URL queue for next pages to be crawled. A second queue is used to store all visited links. If a link is already visited, it will not be crawled again. This functions as a necessity to protect the crawler from deadlock or going in a cycle researching a previously searched site. If a WSDL document is found, it is downloaded and its related web page if available is also downloaded for additional meta-data content about the service. A WSDL document is determined by checking the file extension type (.wsdl), (.asmx?wsdl) or by web content type of text/plain or text/xml and the wsdl page content tags starting to identity a web service description document. The crawler continues by removing a URL in the queue one at a time and processing it. The crawler ends by a preset limit on the number of pages visited or if a desired depth is reached. See Figure 7 for the flowchart of the crawler.

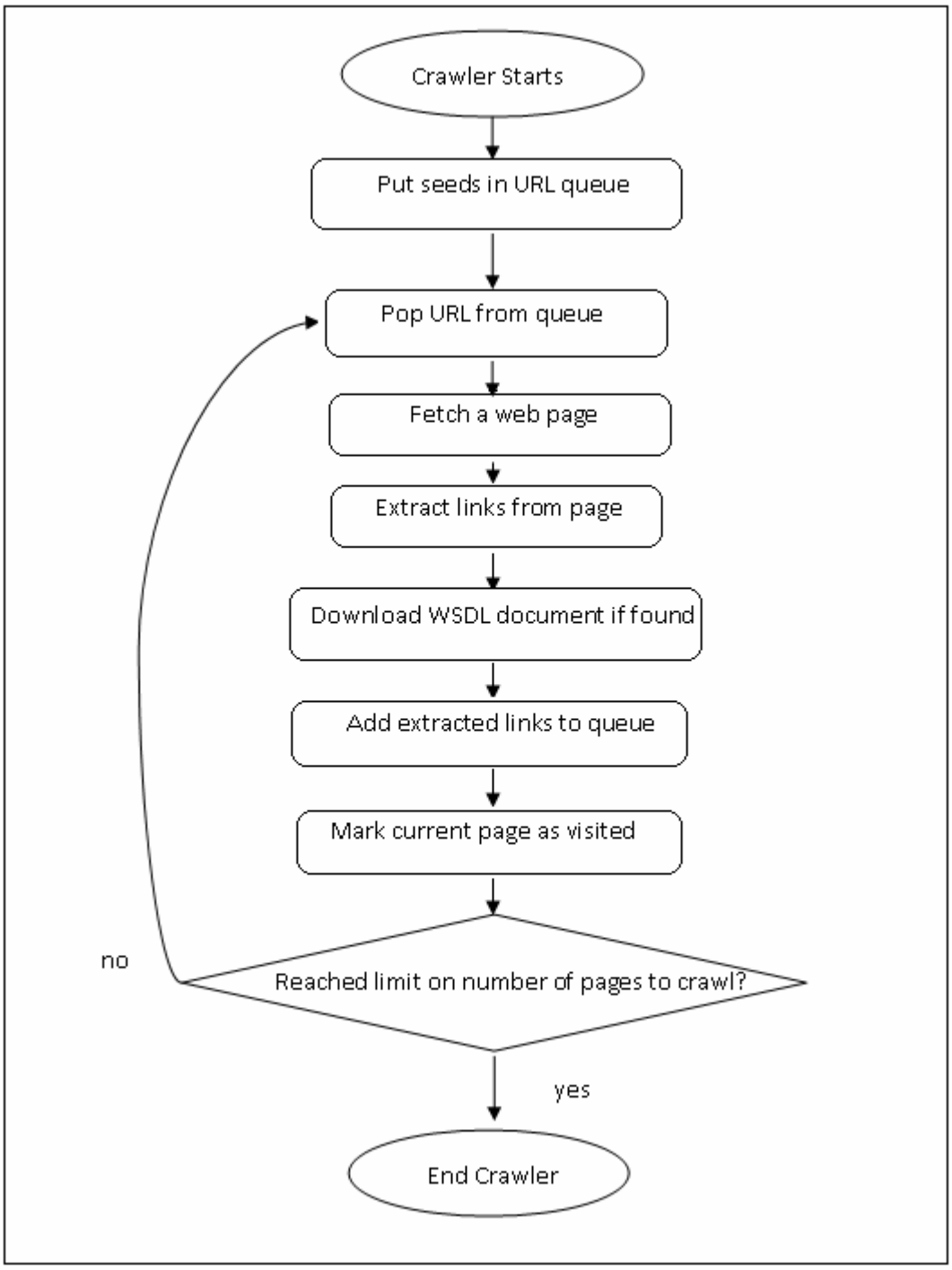

Figure 7: WSDL crawler lifecycle flowchart

The crawler can be made more efficient by using multithreading to speed up downloads in parallel. Due to creating a prototype for the discovery composition framework, having a fast crawler here is not the main function of the overall system. The system does do parallel execution for speedups by running multiple crawlers with different seeds searching for different portions of the web. One of the limitations with using search engines is in its result set limitation. For example, Google only allows a limited number of queries each day and any search query returns only the first 40 hits. The following section shows the modified code for the WSDL crawler to crawl a web search engine. The following example uses Yahoo!'s search API, other search engines are not shown here but uses a similar method as Yahoo's web service with JSON, XML, PHP, or SOAP result sets. JSON stands for JavaScript Object Notation and like XML, is used for data interchange format over the Internet. The PHP output is a serialized PHP data encoding format that is simple text formatting based on PHP data structures. Compared to XML, it is less complex, light weight and platform independent. Yahoo! provides a web service interface into their search infrastructure, in particular REST type web services. REST stands for Representation State Transfer. It uses RPC style operations over HTTP GET or POST requests with parameters encoded into the request URL. Xml or serialized PHP output responses are returned from the request.

The yahoo API is a web service request using the format of Figure 8:

```
http://search.yahooapis.com/WebSearchService/V1/webSearch?
      appid=ID&query=QUERY&output=php
```

Figure 8: Code snippet request to yahoo's web service search API

where ID is the necessary registered application id needed to identify client request for a query and QUERY is a search string. The service provider is located at http://search.yahooapis.com. The web service method is named WebSearchService and the version is V1. The search request follows "webSearch?appid=ID & query=" with a search string and other search options may be applied.

The response result set is returned from the query with the format below, see Figure 9. This result is then parsed for WSDL documents and hyperlink extraction.

The following is an example XML response format

```
<ysearchresponse responsecode="200">
 <nextpage></ nextpage>
          <resultset_web count start totalhits deephits>
                    <result>
                    <abstract>
                              <click_url>
                              <date>
                              <dispurl>
                              <size>
                              <title>
                              <url>
                    </result>
          </resultset_web>
</ysearchresponse>
```

Figure 9: Code snippet response from yahoo API request

The follow PHP code snippet, Figure 10, is an example of a simple web service request using the Yahoo! API service to query its search engine and returns serialized PHP results. The results can be further parsed to get extract the needed contents.

```
<?php

   $appID="your application id key goes here";
   $hostName = 'http://search.yahooapis.com/';
   $serviceName = 'WebSearchService/V1/';
   $method = 'webSearch';
   $baseURL= $hostName . $serviceName . $method .'? appid= ' . $appID .'&output=php
              & query=';

   $request = 'http://search.yahooapis.com/WebSearchService/V1/webSearch?
             appid=$appID&output=php&query=inurl%3A%22asmx%3Fwsdl%22';
   $response = file_get_contents($request);

   if ($response === false) {
        die('Request failed');
   }

$phpobj = unserialize($response);
print_r($phpobj);

?>
```

Figure 10: Crawling yahoo search engine by yahoo's search API

A sample of yahoo's response in PHP format is in Figure 11 below.


```
a:1:{
   s:9:"ResultSet";
   a:4:{
      s:21:"totalResultsAvailable";
      s:6:"266933";
      s:20:"totalResultsReturned";
      s:1:"1";
      s:19:"firstResultPosition";
      s:1:"1";
      s:6:"Result";
         a:10:{
            s:5:"Title";
            s:11:"madonna 118";
            s:7:"Summary";
            s:18:"Picture 118 of 184";
            s:3:"Url";
            s:74:"http://www.celebritypicturesarchive.com/pictures/m/madonna/madonna-118.jpg";
            s:8:"ClickUrl";
            s:74:"http://www.celebritypicturesarchive.com/pictures/m/madonna/madonna-118.jpg";
            s:10:"RefererUrl";
            s:79:"http://www.celebritypicturesarchive.com/pgs/m/Madonna/Madonna%20picture_118.htm";
            s:8:"FileSize";
            s:5:"40209";
            s:10:"FileFormat";
            s:4:"jpeg";
            s:6:"Height";
            s:3:"700";
            s:5:"Width";
            s:3:"473";
            s:9:"Thumbnail";
               a:3:{
                  s:3:"Url";
                  s:41:"http://scd.mm-b1.yimg.com/image/500892420";
                  s:6:"Height";
                  s:3:"130";
                  s:5:"Width";
                  s:2:"87";
                   }
            }
      }
}
```


Figure 11: Yahoo's query response

By decoding the serialized data into PHP structures, you get array structure which can be easily parsed in PHP for data content extraction, see Figure 12.

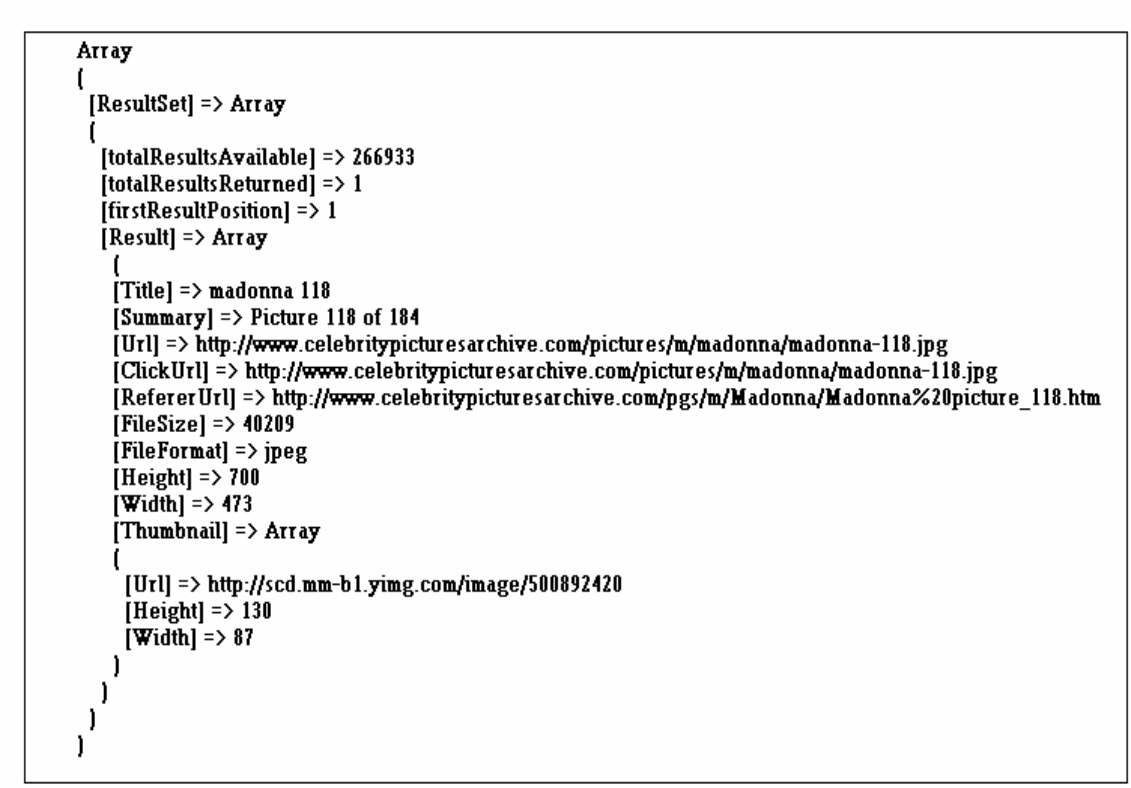

Figure 12: Converting serialized data into PHP arrays

Using the focused crawler, 2864 web services were downloaded, of those, 2525 were active web services collected for the data set in creating the web service repository for the latent semantic document data set.

5.2.2 WSDL Document Data Extract

The next step in being able to search for relevant web services in the repository of web service documents is to create a document index. The primary goal of the document index is for speed ups and optimizing performance. Instead of using a brute force method

in searching all documents in the database against a match in a search query, the aid of an index in a search can greatly improve performance by cutting a portion of the search field. With the given index, the discovery search system can perform searches efficiently and can find documents that are similar to search query terms instead of straight keyword matches that is performed by the original UDDI. There are many present day Information Retrieval methods in creating an index. The method this paper will focus on is a technique using latent semantic analysis (LSA). By using the LSA build matrices as the indexing technique, documents of a query and other similar web services can be retrieved easily by document similarity.

Before beginning any kind of natural language processing technique, a preprocessing step is necessary to remove unimportant words from a document before any text analysis is used on the data. It is used to create a flat list of content words that has semantic meaning representing a document. The collection of WSDL documents is first filtered to remove duplicate web services by service matching. Each file is then preprocessed to extract a flat list of words that best represents the web service. The preprocessing steps are included in Figure 13. The list follows the same general techniques used in traditional information retrieval for TF/IDF and LSI with only slight modifications to work with WSDL structured files to how data may be extracted [13].

1. Validation- confirming that the WSDL document file is in the valid format
2. Text extractor- extract all keywords from the document
3. Content filtering- all symbols, punctuations are removed from the document, only tokens or words are kept.
4. Tokenization- break phrases, part names apart into further tokens using a camel case,name separator
5. Removal of stop words from stop list, common words that are not important are removed from the wordlist; this includes articles, prepositions, common verbs, adjectives and other common words that appear frequently in text documents but do not bring any meaning or help distinguish documents apart.
6. All characters in the documents are converted to lower casing
7. Word stemming- using Porter's algorithm to remove word stems of words that are the same but in different English grammar
8. Remove duplicate in the compiled word list.
9. Discard any words that appear in all documents and discard any words that appear in only one document.

Figure 13: Preprocessing steps of a WSDL document [13]

5.2.3 Latent Semantic Analysis

The choice for using Latent Semantic Analysis in this paper is due to the unknown data we are dealing with. As Latent Semantic Analysis attempts to uncover latent relationships among documents based on word co-occurrence, it can aid search queries in finding similar documents that match the search input by a similarity calculation. The way LSA works is that, for example, if document A contains the terms (cat, dog) and document B contains (dog, cow), it can be inferred that there is a common relationship between documents A and B. LSA does this by using a decomposition step of Singular Value Decomposition (SVD) on the term-document matrix A into three different matrices (U, S and V). The resultant three matrices are "reduced" and a new matrix rebuilds from the reduced matrix. Because of the reduction, noisy relationships

are suppressed or dropped during dimension reduction and relations between documents get merged resulting in similar documents being closer together.

After the previous steps of data extraction and preprocessing, a flat list of content words with semantic meaning per document is then used to construct the vector space model. The vector space model defines documents as vectors or points in a multidimensional Euclidean space. The VSM model is an algebraic model for representing the text documents used for information retrieval. In the VSM model, a document is conceptually represented by a multidimensional vector or points in a multidimensional Euclidean space. The axes or dimension are represented as terms, see Figure 14. The Vector Space Model uses terms to represent both queries and documents. By employing basic linear algebra operations, global similarities between them can be calculated. There are many different ways to model documents for retrieval. Of these models, the Vector Space Model is the simplest to use and in some ways most productive [17]. The vector space model consists of three basic versions of representation: Boolean, term frequency (TF), and term frequency-inverse document frequency (TF-IDF). In the Boolean, vectors are represented as 1 or 0 and document ranking is not possible. In the term frequency approach, the points are represented as the term counts and usually normalized with the document length. The keyword terms extracted from the document can be further associated with a term weight. The TF-IDF considers local and global features of a particular term to the document. In this paper, the focus will be on TF-IDF as the document representation. The TF-IDF value represents the weight of each term to

the document and document collection as a whole. This weighting process normalizes the document set as a whole for a more evenly distributed set.

For example, VSM can be described in a simple example as follows in Figure 14. The table shows the number of terms found in each document. Using the VSM, each document is represented as a vector in the 3-Dimentional space. It can be seen that certain documents bunched together represent their similarity to the number of terms of apple, banana, or car in the document. The coordinate system shows a term space in 3-dimension of the terms apple, banana and car. Each point on the graph represents a document that contains a certain number of the terms apple, banana, and or car. It can be visualized from the graph that document 1 and 2 are closely related with similar apple and banana terms forming a similarity cluster, whereas document 3, 4, and 5 are clustered together meaning they are more related to one another having car as a relationship. Relevant documents in the term vector space can be identified via simple vector operations.

Document / Term	Apple	Banana	Car
Document 1	5	1	0
Document 2	4	2	0
Document 3	2	0	4
Document 4	1	0	5
Document 5	1	2	3

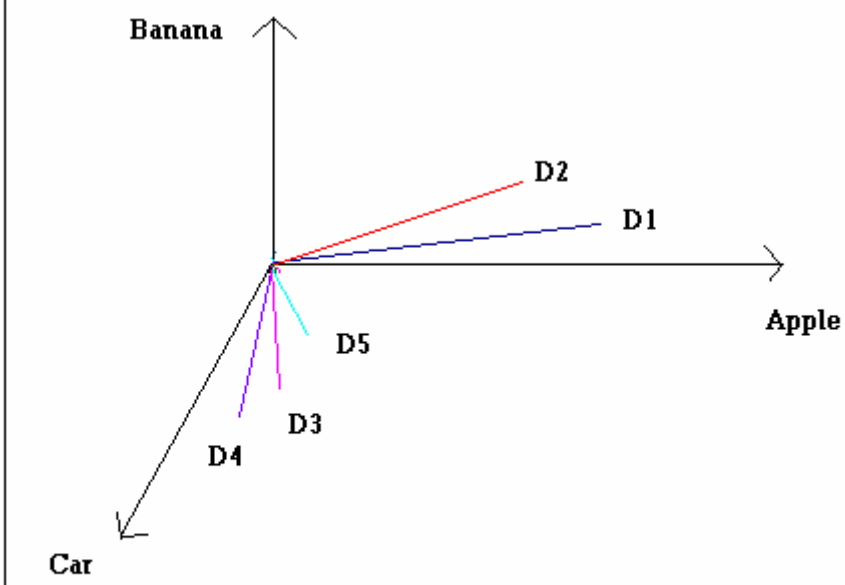

Figure 14: Example of vector space model

In building the VSM, it consists of a matrix with columns as documents and rows as terms that pertain to each document. The matrix is then populated with term weights. The term weights represent the importance of the keyword terms in the document and

within the whole document collection. The weight used here is the inverse document frequency (TF-IDF) weight model using local and global parameters. The weight vector for document *d* can be described in the following equation of Equation 1:

$$\mathbf{v}_d = \left[w_{1,d}, w_{2,d}, \ldots, w_{N,d}\right]^T, \text{ where}$$

$$w_{t,d} = \mathrm{tf}_t \cdot \log \frac{|D|}{|\{t \in d\}|}$$

Equation 1: TF-IDF weighting [17]

where *tf* is term frequency of term t in document d representing a local parameter, $\log \frac{|D|}{|\{t \in d\}|}$ is inverse document frequency (*idf*) representing the global parameter. | D | is the total number of documents in the document set; $|\{t \in d\}|$ is the number of documents containing the term t [17].

The reason behind the weights is that it plays a role in affecting how important data is in the representation of a document. Given the term frequency alone, if the *tf* value is high, then the weight is high. This means that terms that occur frequently in a document would weight more and its importance about what the document is about. This has adverse effects on long documents when longer documents tend to repeat a term more. Also, it is known to spammers to use the same keyword term to rank their pages higher for search engine spams. Due to these scenarios, the global term *idf* is introduced

to weight out the effects and attempts to smooth out the frequency of a word across documents. If a word occurs in more than one document, it means that it is less "precise" and hence its value should go down. Once the term weights are determined, the term-document matrix can then be factorized using singular value decomposition to reduce the space and reveal latent relationships between documents. SVD will project this large term-document multidimensional vector space into a smaller number of dimensions. In doing so, terms that are semantically similar get merged together into a concept allowing computations to go beyond just standard keyword matching. Searching a document in the collection will provide other documents that may be similar but does not contain the original search terms. Finding similarities between these documents then becomes a vector distance calculation.

5.2.3.1 Singular Value Decomposition (SVD)

The fundamental mathematical construct behind LSA is the singular value decomposition. This means that the original matrix is replaced by another matrix that is as close as possible to the original matrix with a column space that is only a subspace of the column space of the original matrix. By reducing to a low rank approximation of a matrix, it is removing extraneous information or noise from the dataset it represents. The decomposition is defined below in equation 2. It breaks a term-document matrix A into three matrices U, S, and V.

$$A = U * S * V^T$$

$$A_k = U_k * S_k * V_k^T$$

Equation 2: SVD [18]

Here U is the orthogonal matrix whose column vectors are called the left singular vectors of A, V is the orthogonal matrix whose column vectors are termed the right singular vectors of A, and S is the diagonal matrix having the singular values of A ordered decreasingly along its diagonal. The reduced matrix is written as Ak from the original by the first k singular values.

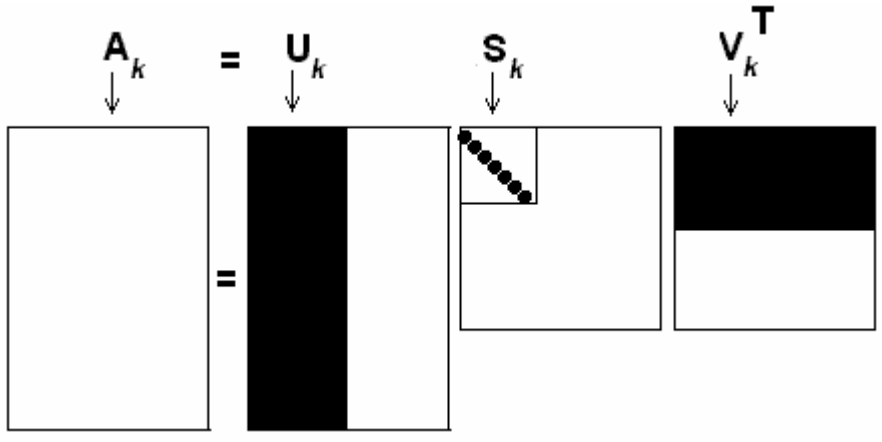

Figure 15: Reduced matrix A from original by the first k singular values[18]

So for a 2 dimensional reduction, the decomposition corresponds to the first 2 columns of V, first 2 rows of U and the square matrix of S in 2x2, and for a 3 dimensional reduction, it is the same way. K is the representation of rank reduction value. Document to document similarity can also be obtained by comparing rows in the matrix VS and word-word similarity can be measured by comparing row similarity in the US matrix. The goal is to find the best rank *k* approximation of **A** that would improve retrieval. The selection of *k* and the number of singular values in **S** to use is still an open area of research [18,19].

Once the SVD is completed, a similarity search measure can be taken on the matrix A. Based on matrix A, similarity search is used to find documents similar or related to a given document. Once a relevant document is found, a larger collection of possible documents may be found by retrieving documents that are similar to the given document. The cosine similarity measure is used to calculate the document to document similarity factor. Given a document *d* and a collection *D*, the problem is to find a number of documents $di \in D$ which have the largest value of cosine similarity to *d*, the maximum value of the dot product *d.di* [2]. More details will be discussed in section 5.2.5.

5.2.4 Web Service Composition

The process of composition is based on creating a new service of out pre-existing web services. Before discussing the composition algorithm, a preprocessing step is necessary to extract the required data from the WSDL document to build the input/ouput indexes for composition. The composition preprocessing steps requires parsing the WSDL for input and output parameters. The <service> element is first located for the service name, then the <binding> element that matches the service is searched to get the set of operations that the <binding> element includes, next <portType> element that the <binding> element points to is used to get operations. Once we find the <operations> of the <portType> element, we can retrieve the input/output message declaration that is associated with them. The input/output parameters of an operation use a message as their type. An operation may either have an input or output or both. A <message> is further defined as a collection of parts. Each part represents one thing to be sent or received. The message part uses the XML schema to define its part's type. A message part can be

a complex type, element pointing to a named complex type or build-in types. The input/output parameter tree for each service is created for visual display in the GUI interface. See Figure 16 for a diagram of the order in parsing steps in retrieving the input and output parameters for each operation in a web service document.

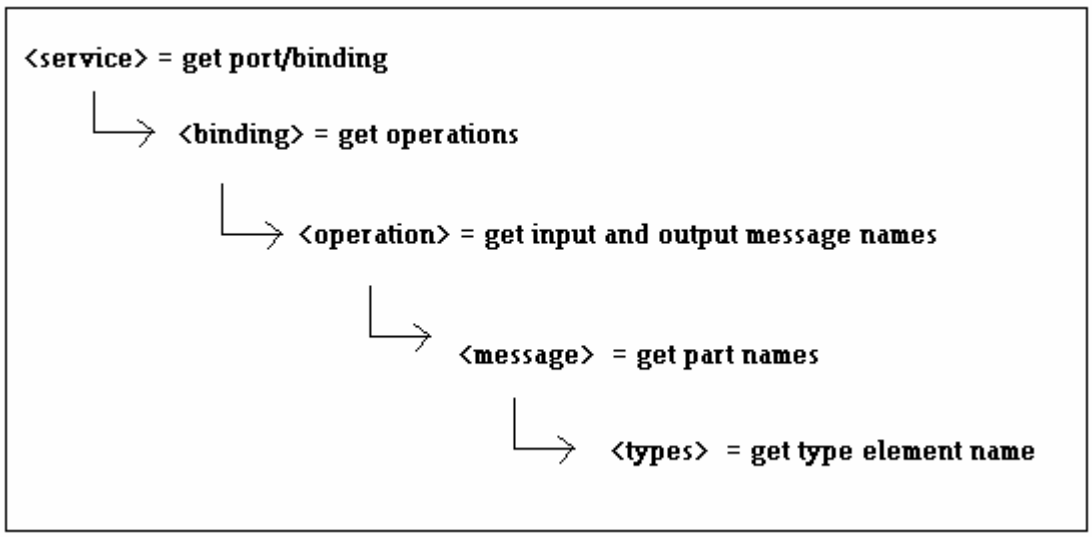

Figure 16: Order of parsing WSDL document for operations and In/Out parameters

The message names, part element names and type names are used for composition matching. Each name pertaining to the service will be processed using porter's stemming algorithm to reduce a word to its root form and also the removal of camel casing in a word. This method is used to broaden the matching in real world services since service providers may have named message parameters using camel casing or used different naming schemes.

The algorithm used for the composition framework is one based on BITS algorithm [15]. Slight modifications are made to the algorithm for the framework to provide partial composition when full compositions are not found and a graphical tree structure for visualization. The algorithm uses back to front searching with the pre-computed input and inverted output indexes. These indexes create the searching lookup links for the web services. The algorithm runs creating a tree structure for possible composing candidates, once a potential composing path is found, the compiled tree is displayed. The following is an example showing five web services of a small repository set, see Table 1, and how the input and inverted output index is generated for the algorithm, see Figure 17. Figure 18 shows the algorithm with given search parameters R.o as required outputs, and R.i is the required inputs. The algorithm in basic terms builds a backward output input tree as it performs the back to front search. It starts from the given R.o parameters and searches its way back to the R.i inputs. The search begins by looking up the web services that can produce as outputs the set R.o. We collect a set of web services that can produce 1 or more outputs that is a subset of R.o. Given the collection, we take the Cartesian product of List n to get all Cartesian sets. With the Cartesian sets, we choose the minimum set as the precondition set. By using only the minimum set, we reduce the search time and also the minimum number of web services that are required to produce the given required output R.o. Once we find the precondition set, we check each element in the set to see if it satisfied as one of the R.i requirements. If an R.i element is found, we end the node as leaf, otherwise, we check if the current web service contains any preconditions. The precondition of the next node connection is found by first looking up the inputs for the web service and then taking the precondition

step for those inputs. The tree building process continues until no more preconditions are found or a branch limit is reached. The branch limit is set based on the number of connections of web services wanted for a composition tree.

Web Service	Inputs	Outputs
WS1	ZipCode, District	Weather, Address
WS2	Zone	ZipCode, Province
WS3	Road	Temperature, District, City
WS4	ZipCode, City	Weather, Address, Temperature
WS5	Longitude, Latitude	City

Table 1: Web Services Repository

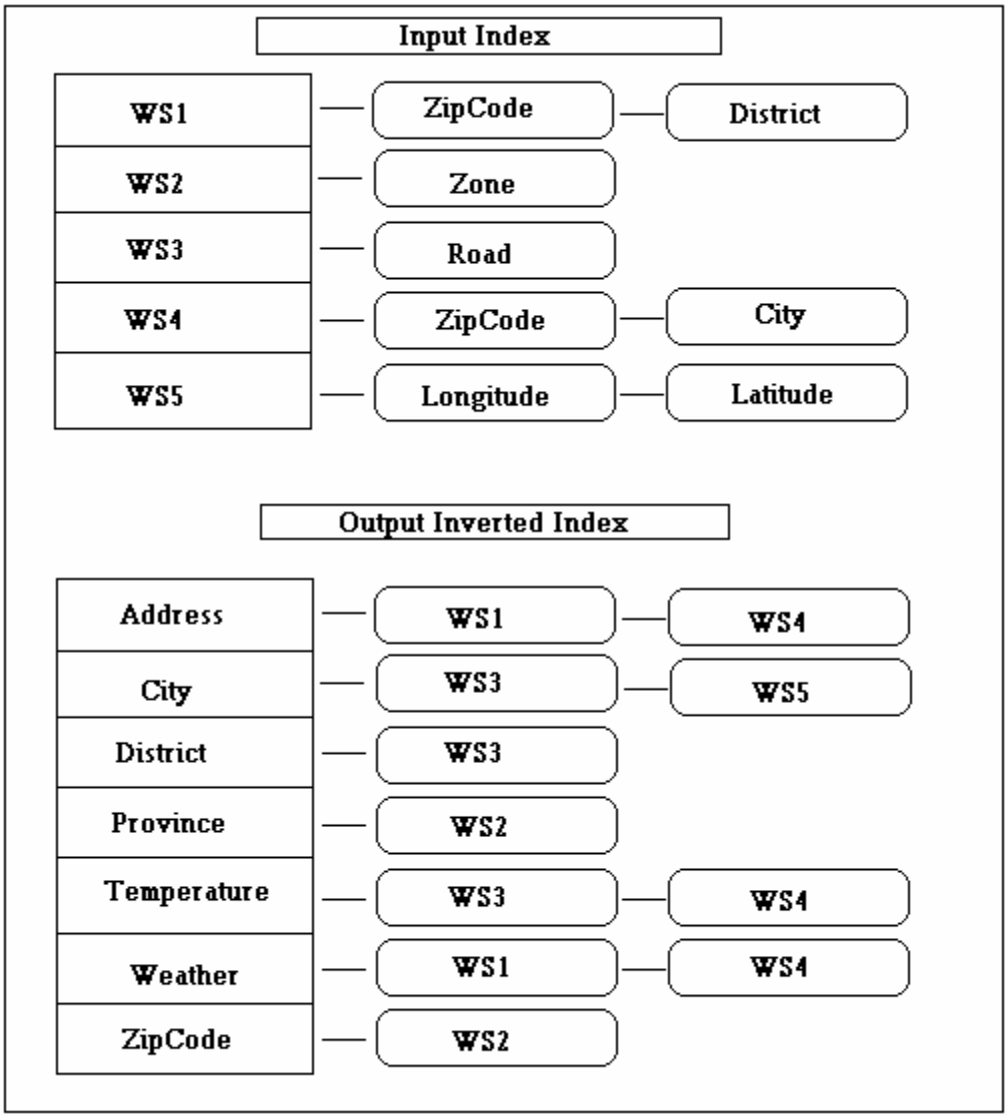

Figure 17: Example of Input and Output Inverted Indexes

1) Get R.o parameters P1, P2, ... Pn and create root node
2) Look up from output inverted index for List L1, L2, ... Ln for each Pn
3) Interpreting Lists L1, L2, .. Ln as relations, and then performing cartesian product for these n lists giving sets 1... set n
4) Each cartesian set is a path, use set1 .. setn as preconditions for each path node For all cartesian set found, get the minimal sets as preconditions
5) For each element in set i, create and insert child node,
   check if its R.i is found, if found end node as leaf,
   else:
   Look for element i of set i for its precondition and repeat steps 2-5,
   First lookup the inputs of the web service, and then find its precondition set to produce those inputs
   - check that ending leaf node is a R.i
   - if not R.i then partial tree is captured
   - full path is complete when all initial R.i is satisfy at end nodes

Figure 18: Composition algorithm

A sample output tree is displayed using the algorithm for a query composition given R.o = {weather, address, province}. The R.i = {Road, Zone}. The composition tree shows two possible full paths and 1 partial path. The sided nodes form the full paths in the diagram. The full path 1 consists of R.o :{WS1, WS2}-> WS1: {WS2,WS3}, full path 2 consists of the path R.o:{WS4, WS2}-> WS4:{WS2, WS3} and the partial path is given as R.o: {WS4, WS2} ->WS4:{WS2, WS5}. See Figure 19.

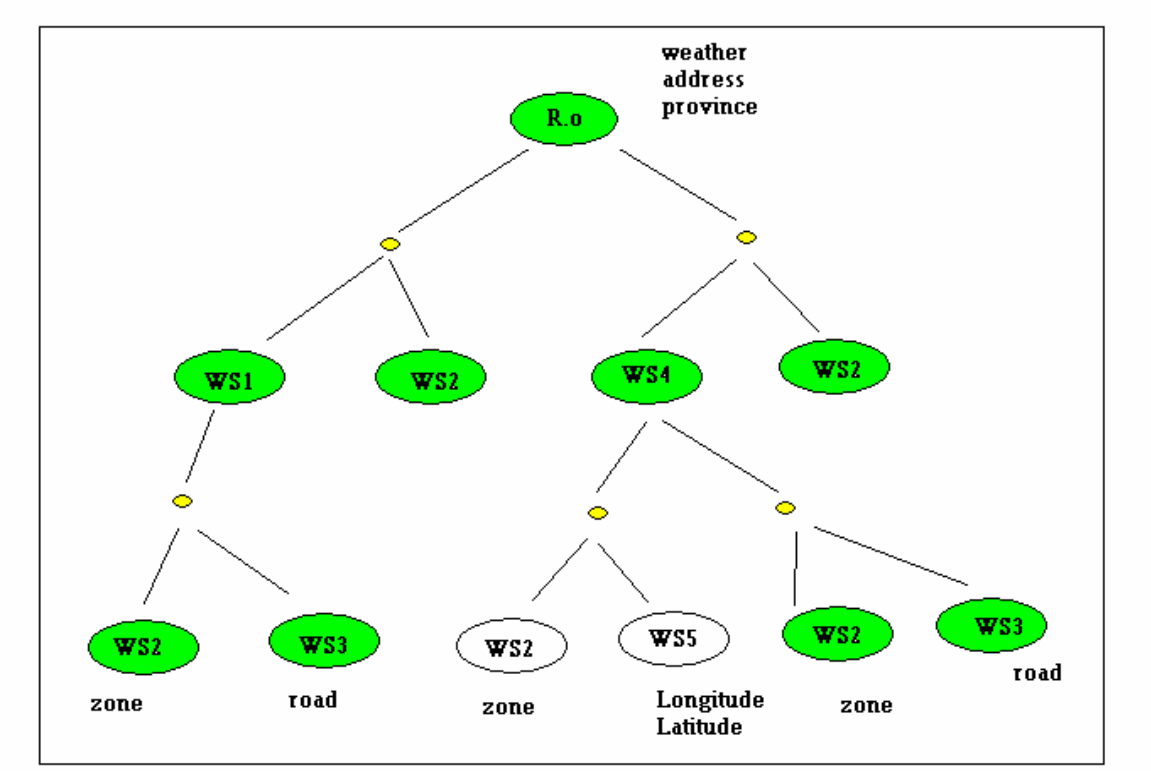

Figure 19: Full Composition Tree Display

A composition tree is given if a full path is found. If only a partial path is found, then the partial path is given to the user to fill in the rest. Each node of the tree represents a web service to creating the connection to the desired input/output requirements. In viewing the graphical display of the tree structure, each node can also be viewed in further details by looking at its full WSDL document, its input/output parameter graph, or other potential similar web services like the current one using the related neighboring graph. A partial tree generated by the framework is shown in Figure 20. This example uses the given R.o = {weather, address}, and R.i = {district, longitude, province}. This

tree is only a partial tree because it satisfied the input district and longitude but not province.

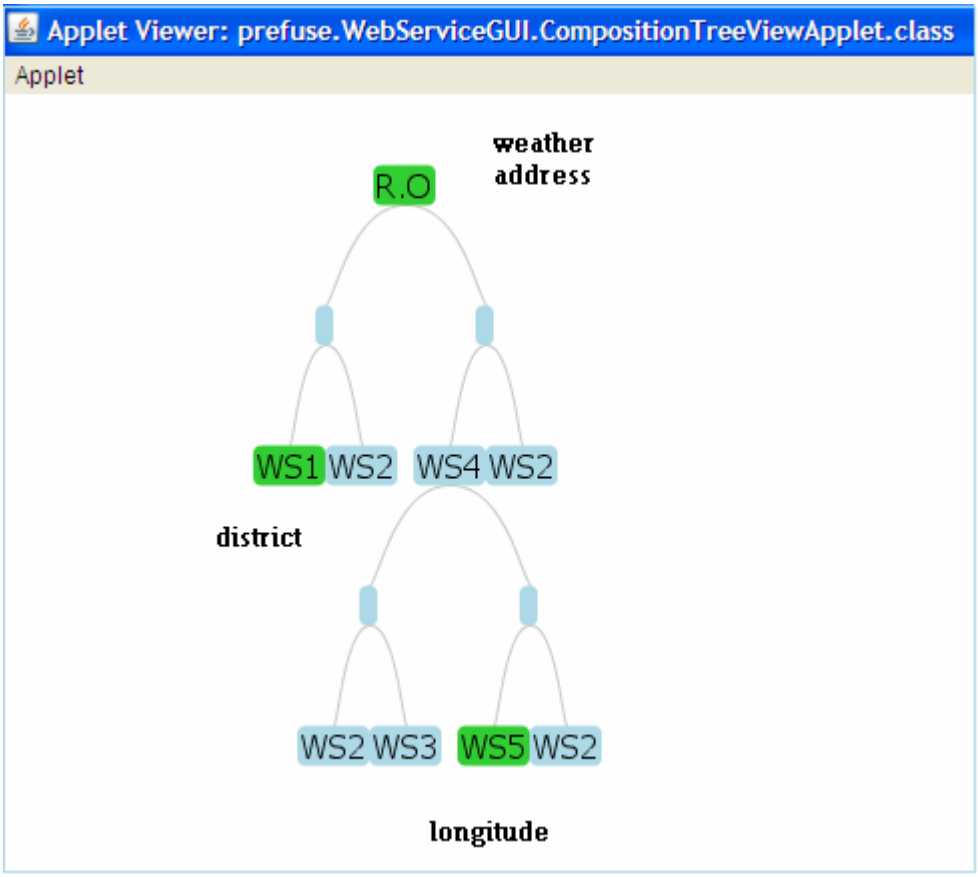

Figure 20: Partial Tree Composition

5.2.5 Query Processor & Finding Similarity in Documents from a Query Search

A user can query the repository database to find relevant document, the results use the calculated SVD matrix representation of the data. The query submitted by the user is formulated as a set of terms with added weights, represented like the document vectors. It is likely that many of the terms are not in the document set, so the vector will contain a number of zero components. Document vectors in the matrix are then matched against

the query vector, finding those geometrically closest to the query according to the cosine similarity measure. The cosine similarity measure is the cosine of the angle between the query and the document vectors [19]. The cosines of the angles between the query *q* and document vectors are computed by comparing a query vector *q* to the columns of the approximation A_k to the term document matrix *A* in Equation 3.

$$\cos\theta_j = \frac{(A_k e_j)^T q}{\|A_k e_j\|_2 \|q\|_2} = \frac{(U_k \Sigma_k V_k^T e_j)^T q}{\|U_k \Sigma_k V_k^T e_j\|_2 \|q\|_2} = \frac{e_j^T V_k^T \Sigma_k (U_k^T q)}{\|\Sigma_k V_k^T e_j\|_2 \|q\|_2}$$

for j=1, ..., d

Equation 3: Cosine Similarity measure to find document similarity [19]

Here, *ej* is defined to be the *j*th canonical vector of dimension *d*(the *j*th column of the *d x d* identity matrix), the *j*th column of A_k is given A_k *ej*

Usually, the query and document vectors are sparse, so the dot product and norms in equation 3 are generally inexpensive to compute. Furthermore, the norms ||Sj||2 can be pre computed once for each term-document matrix and be used for all queries. For further mathematical explanations of the underlying workings of vector space models and SVD, refer to [19].

The role of the query processor is to allow associated WordNet words to the user input terms for additional semantics in broadening the search field. The query processor

will formulate queries and communicate with the document indexer to find comparisons in the most similar documents. It will return all similar documents to the user interface where the results will be displayed.

5.2.6 User Friendly Graphical User Interface (GUI) for WS searching

The GUI search interface provides similarity web service search for users to input keyword query requests in searching for web services. It contains a word suggestion box providing wordNet to associate more search terms in the query. The advance option panel also provides an option to use different pre-computed SVD matrices during the search. The number of results to return and the cutoff value for the cosine similarity measure can also be adjusted. Search results are ranked and returned to the user in a list format with a short description of the service. Each service on the returned list can be clicked to view more detailed information. The interface provides the return list on the left side panel. The right side panel provides visualization of data in three different forms: WS graph view, Full WSDL view, and a quick input/ouput parameter view. The WS graph view shows the current selected web service and its latent web services as connected neighbors on the graph, it also contains a control panel that can customize the graph view by number of connected nodes to display and the distance of view space. Highlighting over the selected service will sub highlight all first level connected nodes to the current selected service. This represents the closest similar service to the once selected. The full WSDL view displays the web service WSDL document and the input output parameter tree view displays the input and output parameters for that service. The

visual graph is shown in the result set for better visualization of result data. See Figure 21, 22, 23, 24 for the different view panels.

Figure 21: GUI interface with WS graph view on highlighted relevant neighbor services

Figure 22: GUI interface with WS graph view

Figure 23: GUI interface with full WSDL view

Figure 24: GUI interface with quick In/Out parameter view

The composition view of the GUI interface is used by clicking the advanced search tab. Once selected, the composition search panel is presented with a search panel on the right side and a tree composition view on the left panel. It provides an entry for input and output parameters, and options to allow for direct keyword matching or similarity match and whether to use wordNet as part of the matching process. When a search is requested, the result will be presented on the left panel in a tree graph. A search box is provided to search the tree in more detail. See Figure 25.

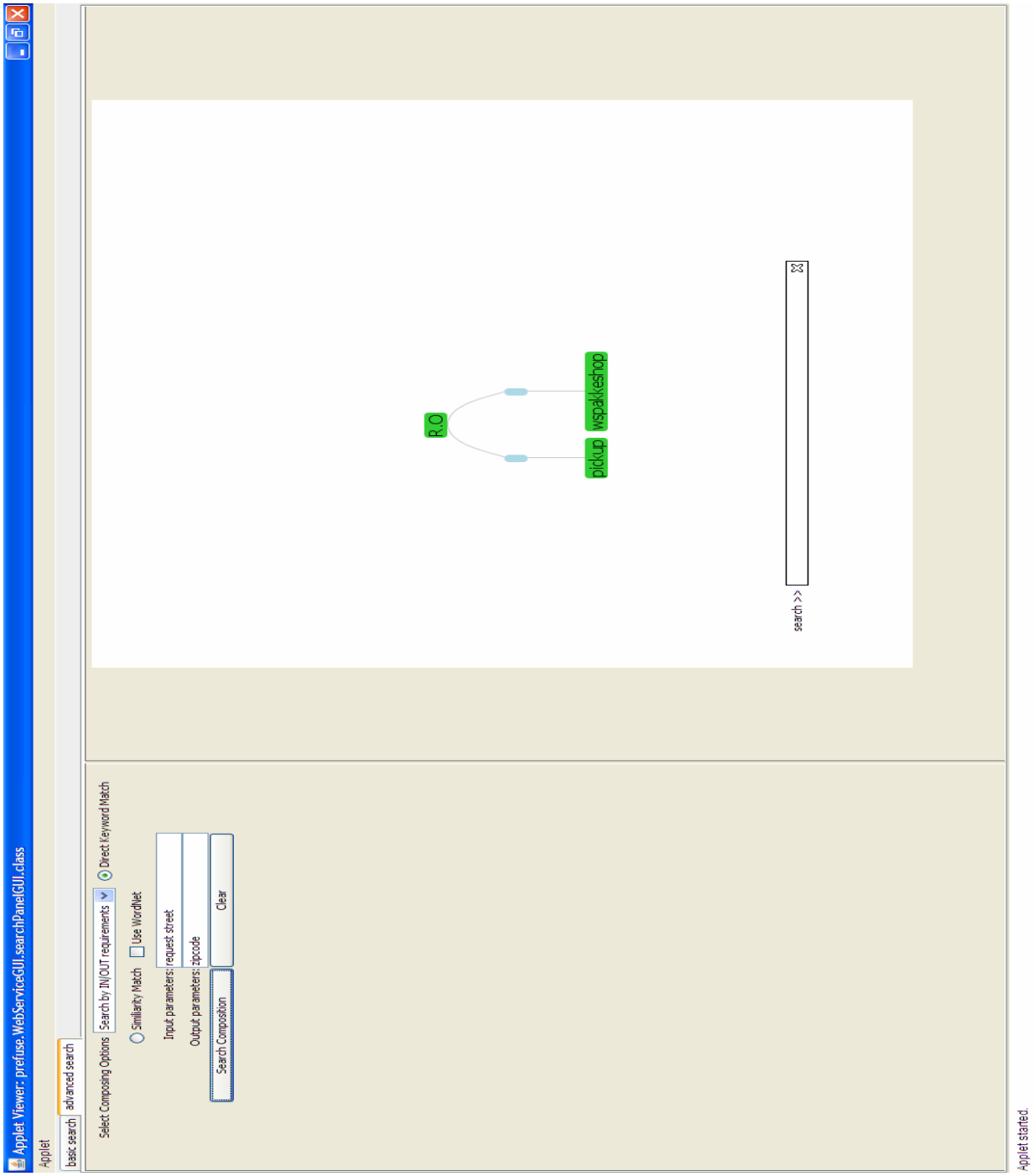

Figure 25: Graphical interface for web service composition

6. Analysis/ Experimental Evaluations

The following section will present the experimental results of the framework that demonstrate the performance of latent semantic service discovery and service composition. Latent semantic based service discovery in general has better results than the general keyword match. The techniques used by the framework are more effective in returning latent semantic results that are hard to match directly from keyword matches.

6.1 Experiment Setup

A set of experiments are done to validate the performance of the framework. Here we show the precision and recall on similarity queries and investigate on the web service compostion method. All experiments on the framework are done with a benchmark of 2525 active web services and ran on an intel CPU at 166GHz, 1G Memory Windows system. The SVD matrix is pre-computed using Linux machines with 64 bit x 86-64 running 32 bit Fedora 9 on AMD CPU.

The graph below shows the computation time of computing SVD as the dimensionality increases. As the term space gets larger, the time increases exponentially, see Figure 26. Eventhough SVD computation can get expensive, it can be precomputed one time, and any additional adds to the matrix can be done using SVD matrix updating or folding-in that does not require recomputation of the SVD matrix. The computation on the experiment of term space 8914 x document size 2525 took approximately 7hrs.

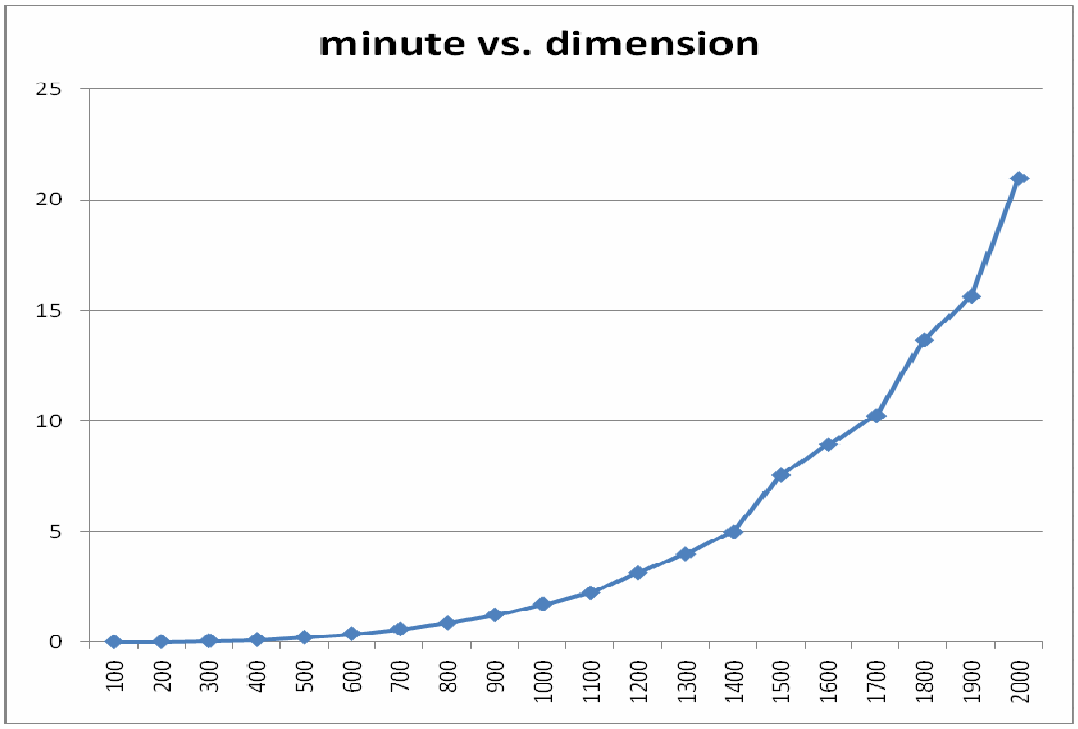

Figure 26: Time computation of SVD matrix

Additional reduced matrices are computed at varied k values for experiments on finding the best k value. Figure 27 shows the Singular matrix values at each rank k. This graph can show the amount of data loss at each level. At 2316, the value is 0.0112. as the cutoff value increases, more data or noise is loss, as terms get merged together. Using the graph, various k rank matrices are generated for testing in finding the best k rank value. Matrices at rank k: 50, 400, 1060, 2000, 2525, in particular, are used for the experiments. The singular value at each of these k values are: 1004.1, 198.99, 51.12, 6.31, 0. This shows that at 2525, no data is loss and the matrix is the same as the original computed SVD. As more is being cutoff, more data is loss and higher dimensional data are being merged into lower dimensions and grouping data together in some way.

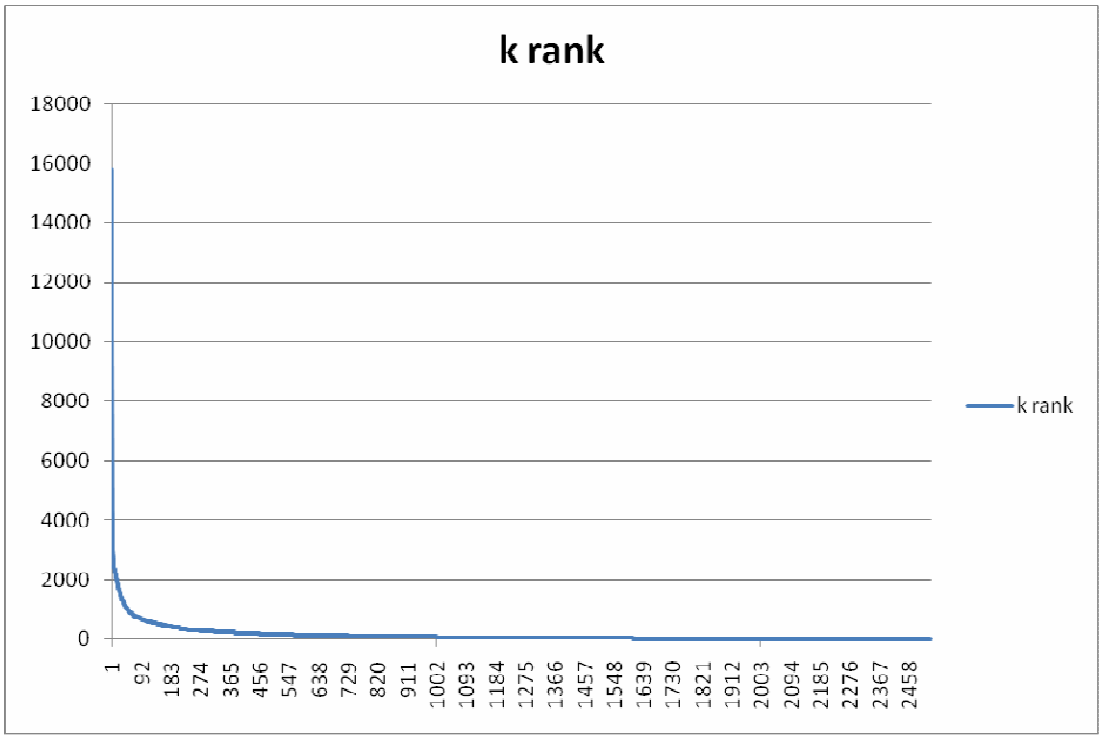

Figure 27: Graph of S matrix values at different k rank (x,y)(data values, k)

6.2 Experiment Study on WS Discovery

The next set of experiments is tested to validate the performance of the framework search queries. The performance is done using precision and recall to investigate between keyword queries and similarity queries to show the contributions of the different components of the system. The precision metric is defined as the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search. It measures the fraction of how relevant the retrieved documents are. The recall is defined as the number of relevant documents retrieved by a search divided by the total number of existing retrieving documents which should have been

retrieved. Recall measures the fraction of how relevant to the query results is successfully retrieved [20].

The experiments compared the framework with the general keyword search similar to that of UDDI. A benchmark of 14 web service operations from a variety of domains with different input/output sizes and description sizes were tried. Figure 28 shows the average recall-precision curve. The framework using the SVD method shows better results than the keyword search method. For example, a search for weather in keyword searches would generally just return results that match the service name with weather. On the other hand, using SVD provides relevant results like temperature, humidity, elevation, wind, and Celsius to Fahrenheit calculation related services that may be useful to what the user is searching for. From the experiments, it can be seen that precision is high at small recalls, but as the search recall increases, the precision drops in the keyword search method while the SVD method maintains a slower drop rate. Figure 29 shows the top-k precision on the search. The top-2, top-5, top10, and top 15 precision of the framework are 100%, 95%, 87%, 84%, respectively, higher than the keyword method by 10-25 percentage points. This demonstrates that using LSA for web service search gives a higher return of results due to its nature to return similar web services.

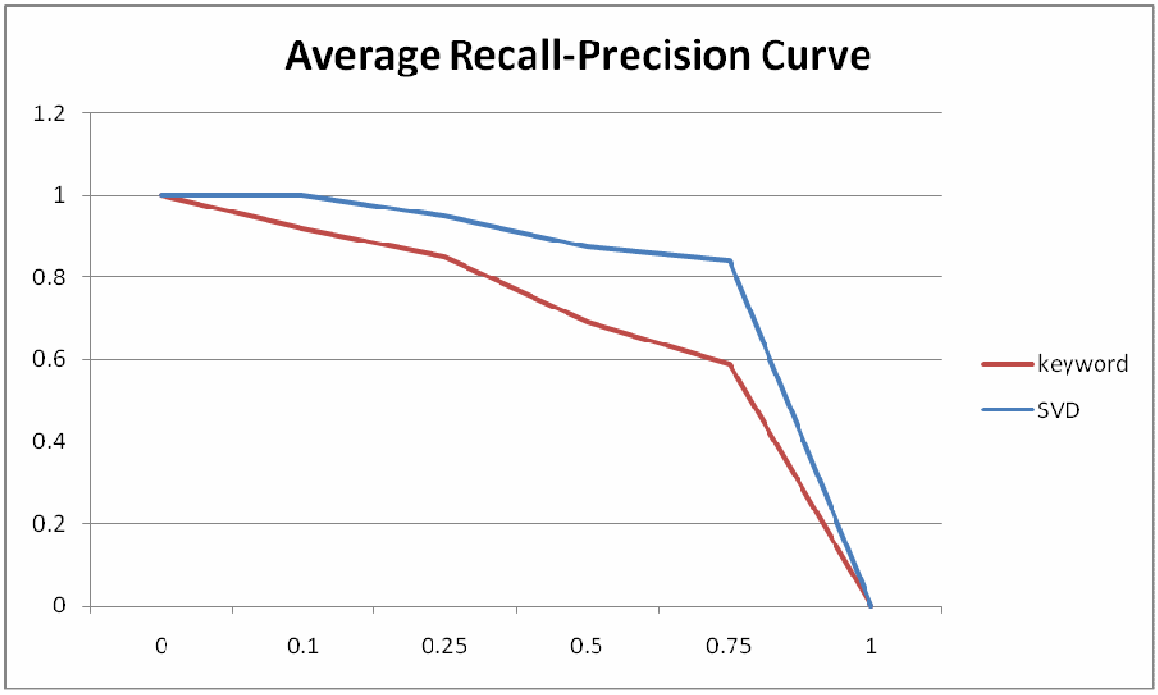

Figure 28: Comparision of keyword to similarity queries

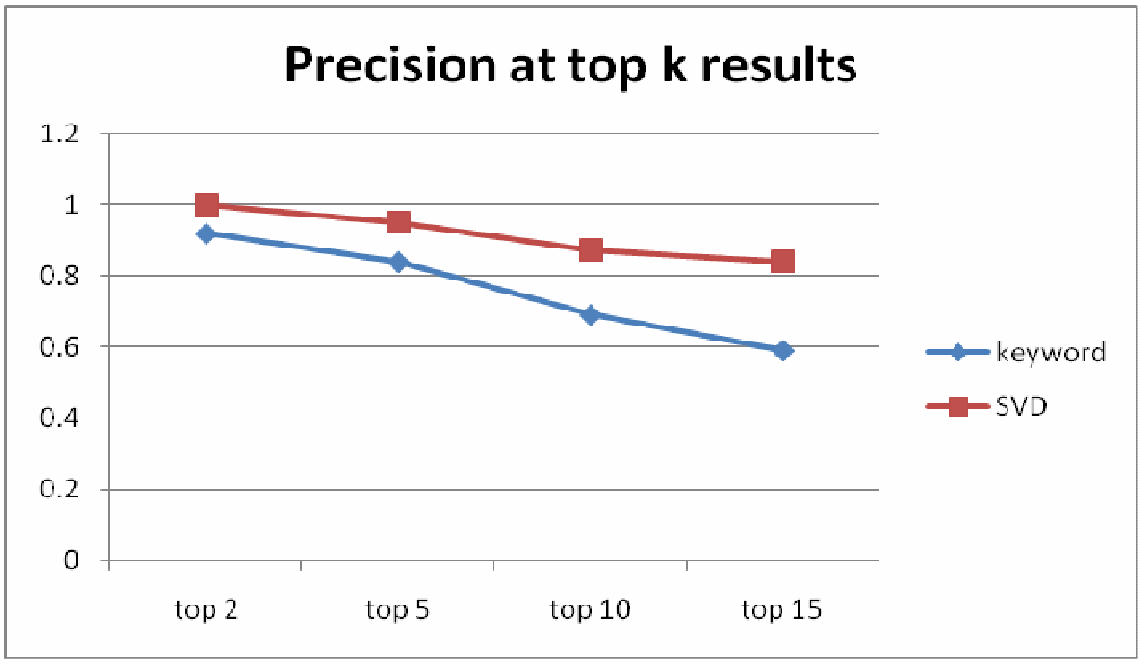

Figure 29: Comparision of keyword to similarity queries for up to the top 15 results

Additional experiments were taken on the framework itself for comparisons. Evaluations were made on finding the best k rank to use for searches. Various k- ranks, 50, 400, 700, 1060, 2000, and 2525 were used to test a set of 10 web service operation search queries. The results in Figure 30 shows the average precision for the top n results returned. The graph shows rank 400 on average would be a good rank to use. When k is 50, there is a significant loss in precision at low recall but seems to average out when more results are returned. This is due to the high cut in the dimensionality, since there is a high data loss for using k= 50, more data is merged into smaller dimension, which is showing the less precise data return. Overall, all other ranks seem to do well at the top 15 results. There is a slight decrease in rank 1060, and 700 after the top 15, while rank 2000, and 2525 drops later at the top 30. Only rank 400 on average seem to maintain at a slightly higher precision rate than the rest after the top 30 results. Figure 31 shows the graph in a different perspective than the line graph of Figure 30, here it can be seen more clearly that rank 400 maintains a slow drop in precision up to rank 200.

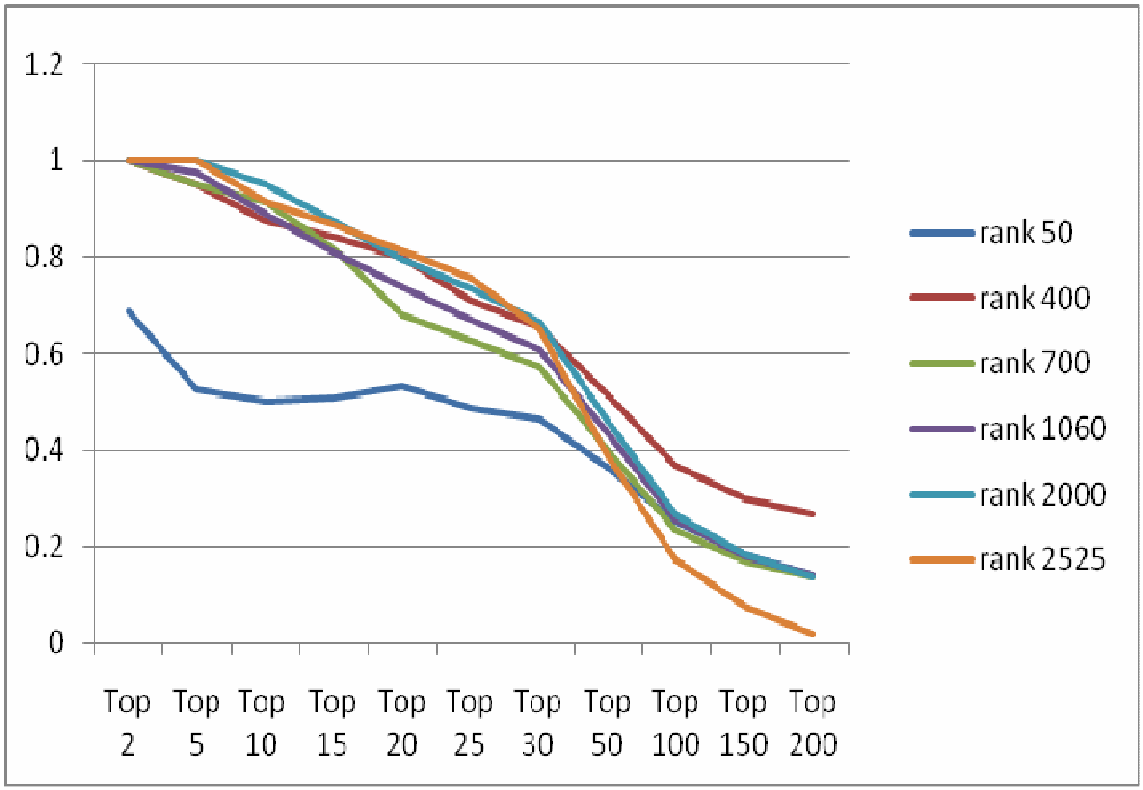

Figure 30: Avg. precision on similarity queries at each top 200 results from different SVD at k rank by curve

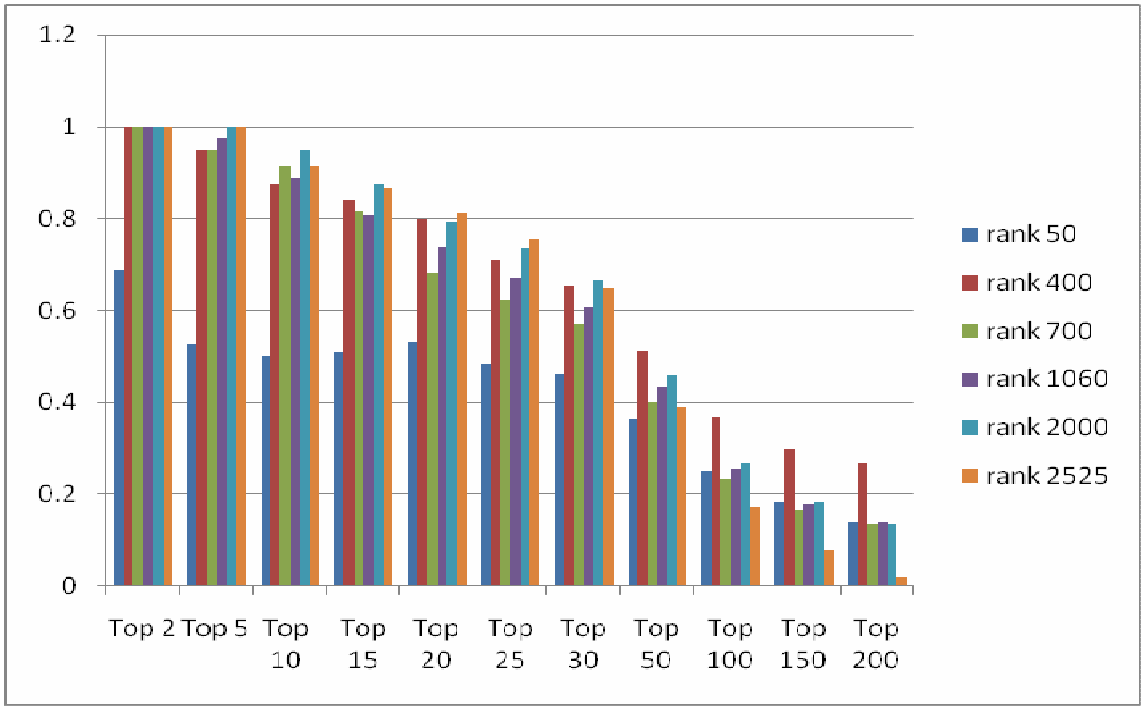

Figure 31: Avg. precision on similarity queries at each top 200 results from different SVD at k rank by bar graph

Applying different cutoff values to the returned result gives a varied recall value. The smaller the cutoff, the higher recall value of web service results returned that are relevant to the search. Figure 32 shows that rank 1060 has a higher recall value at 0.1 cutoff but reduces in value at 0.5. Rank 50 shows that it does poorly for any cutoff greater than 0.1.

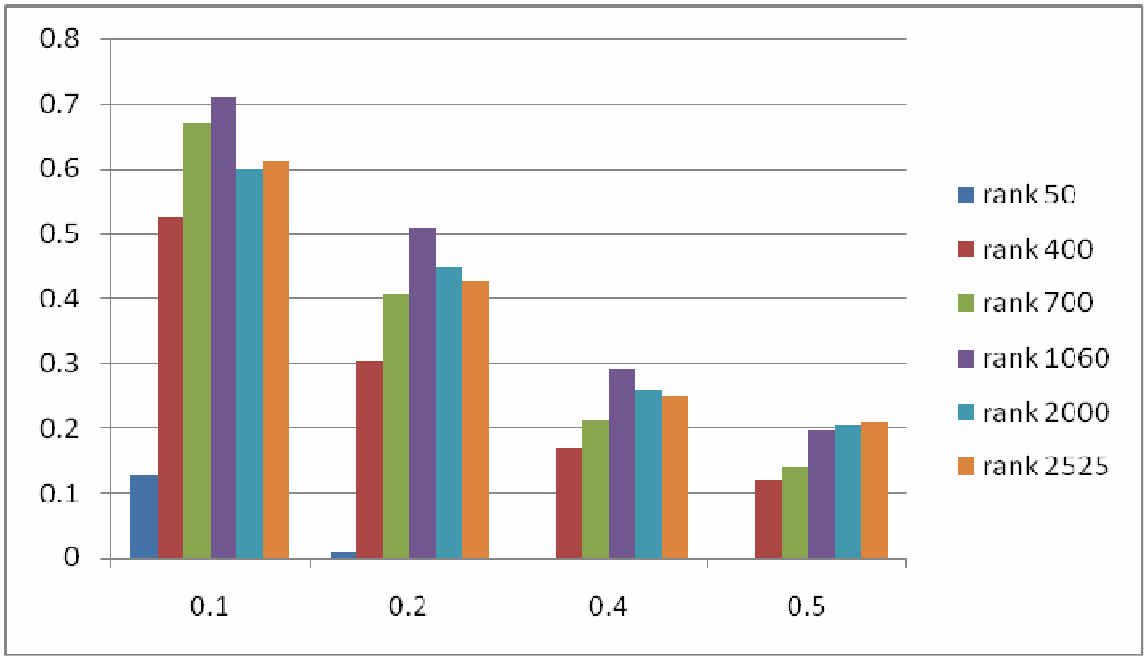

Figure 32: Avg. recall on similarity queries with varied cutoff values of SVD at k rank

Overall, the method of LSA has shown improvements to the keyword search. More relevant results are retrieved from the user's request. It can be seen that different k values also affect the data returned. On average, when k is near 400, the data seems to be at its best.

6.3 Investigation on Composition of Web Services

Composition search in the framework has the ability to build complex services. It returns the desired functionality given input or output requirements. A full composition tree is returned if web services can be bind, otherwise a partial tree is returned for partial composition. The composition can be of any length.

For example, results returned by composition search on input address and output zip give the following composition of web services, as seen in Figure 33. The composing of web services datadoorsauthentication , mpapiwsdl, dotsfaudprotectionlite gives the desired input and output. Datadoorsauthentication takes input address, and dotsfaudprotectionlite returns output zip.

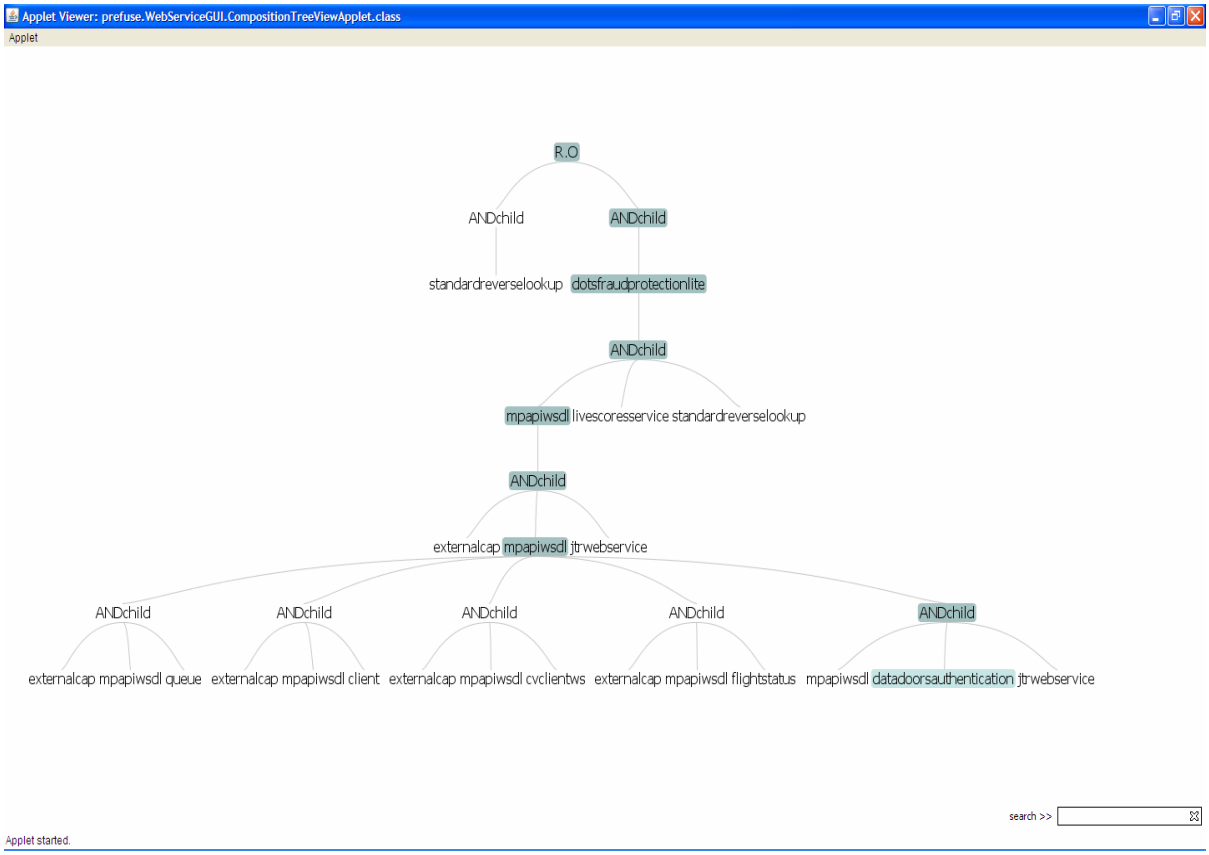

Figure 33: Composition of input address and output zipcode

Another example with two inputs and one output is given as input {request, street} and output {zipcode}, the composition found is a composing of services pickup and wspakkeshop, see Figure 34. Figure 35, 36, 37 shows different composition searches. The last example is one using wordNet to searching for composition. Here, the example shows the required input as road and output as territory. The found web service best matching these requirements are web service WS3 with road as input and output as temperature, district and city. Here the use of wordNet matched territory to district and city as its similarity. In these examples, what are highlighted in green are the exact

matches to the required input and output parameters. Each node is a web service, and a blue node means a partial path given requiring necessary services in order to complete the requirement parameters.

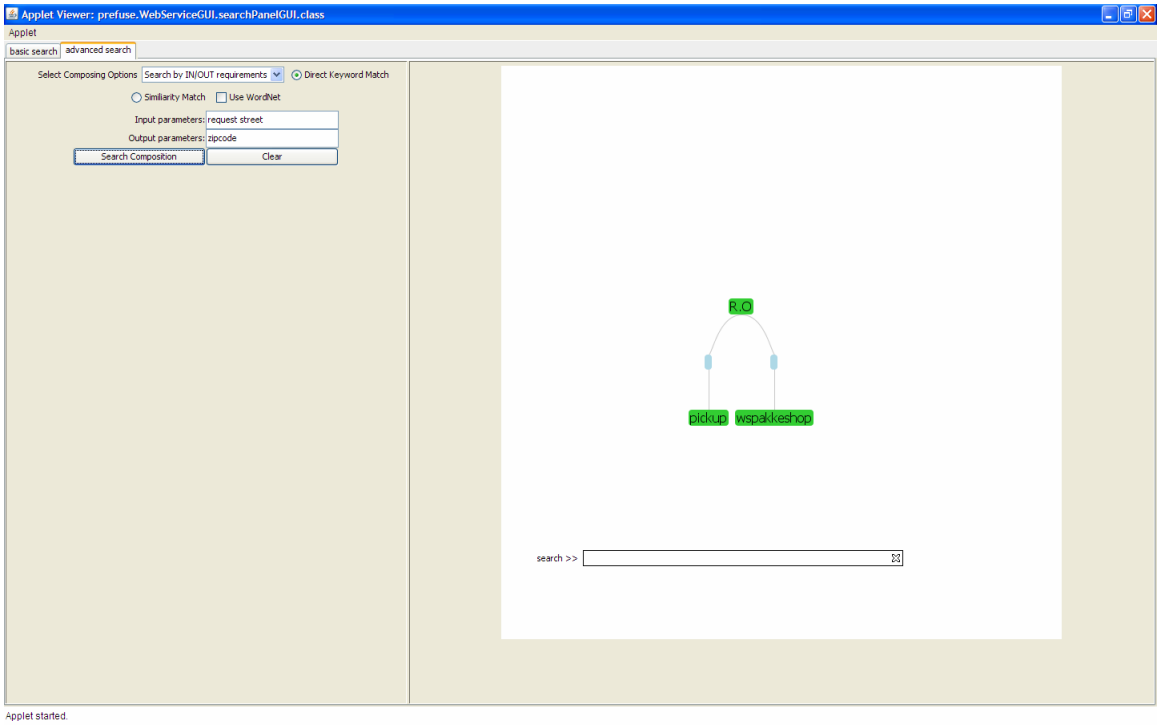

Figure 34: Composition of input request, street and output zipcode

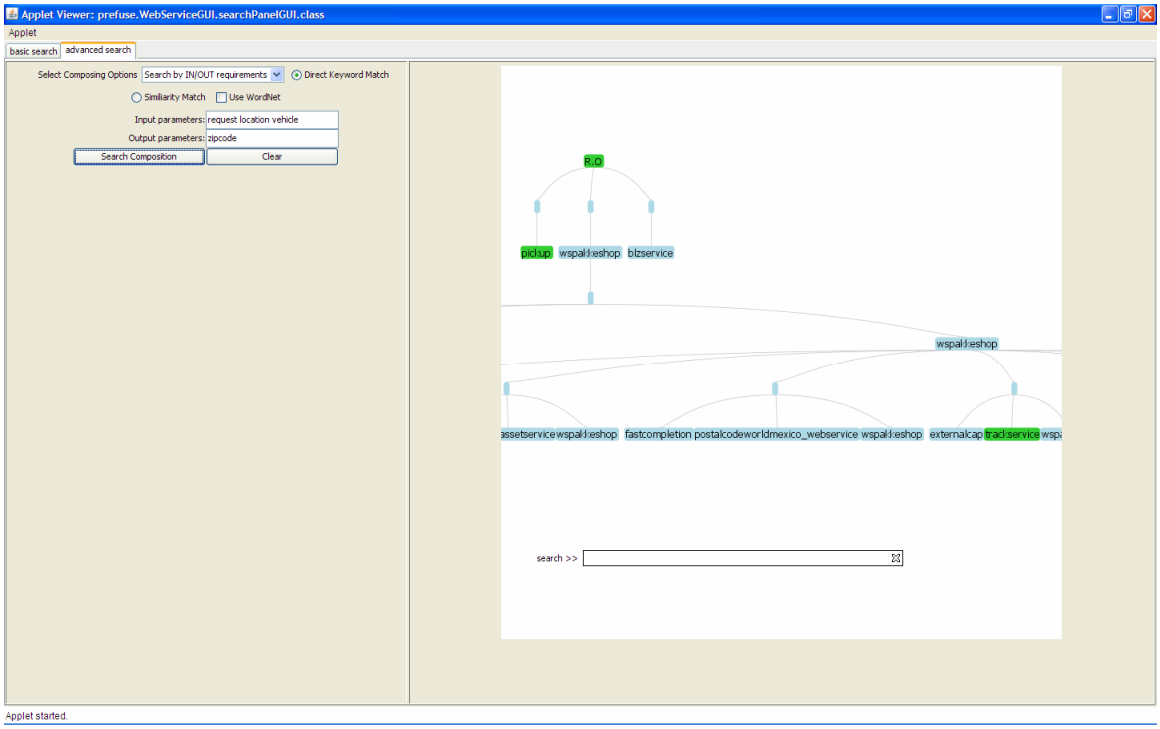

Figure 35: Composition of Input: request, location, vehicle Output: zipcode

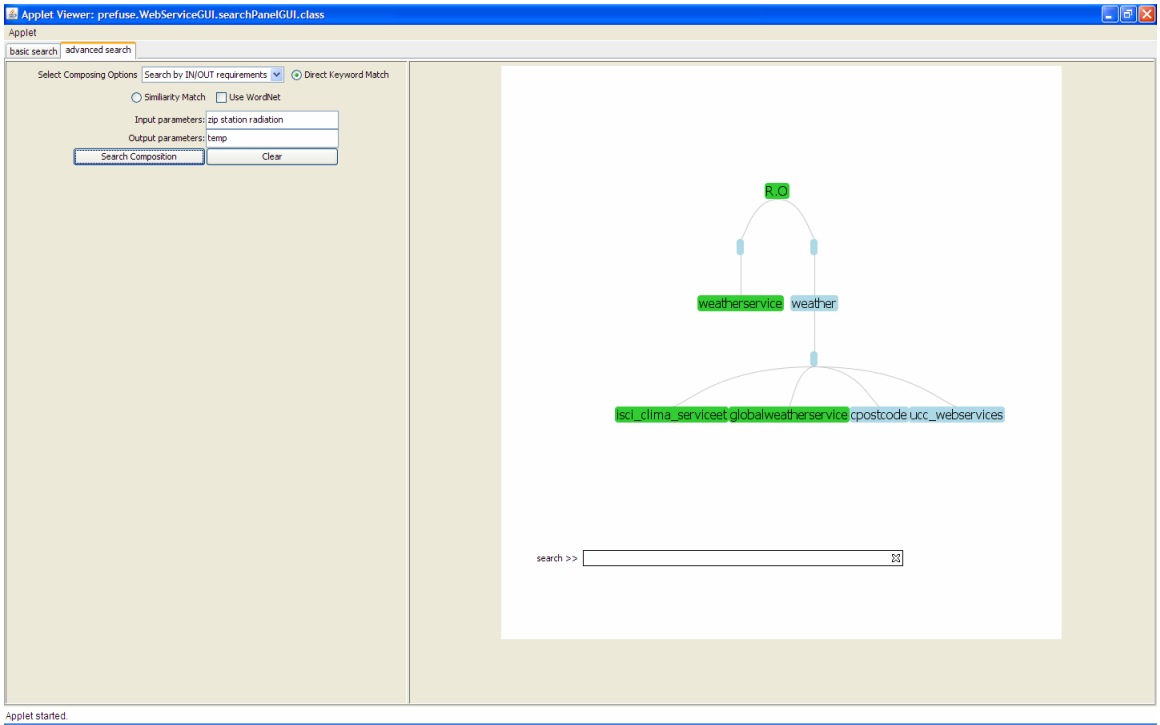

Figure 36: Composition of Input: zip, station, radiation Output: temp

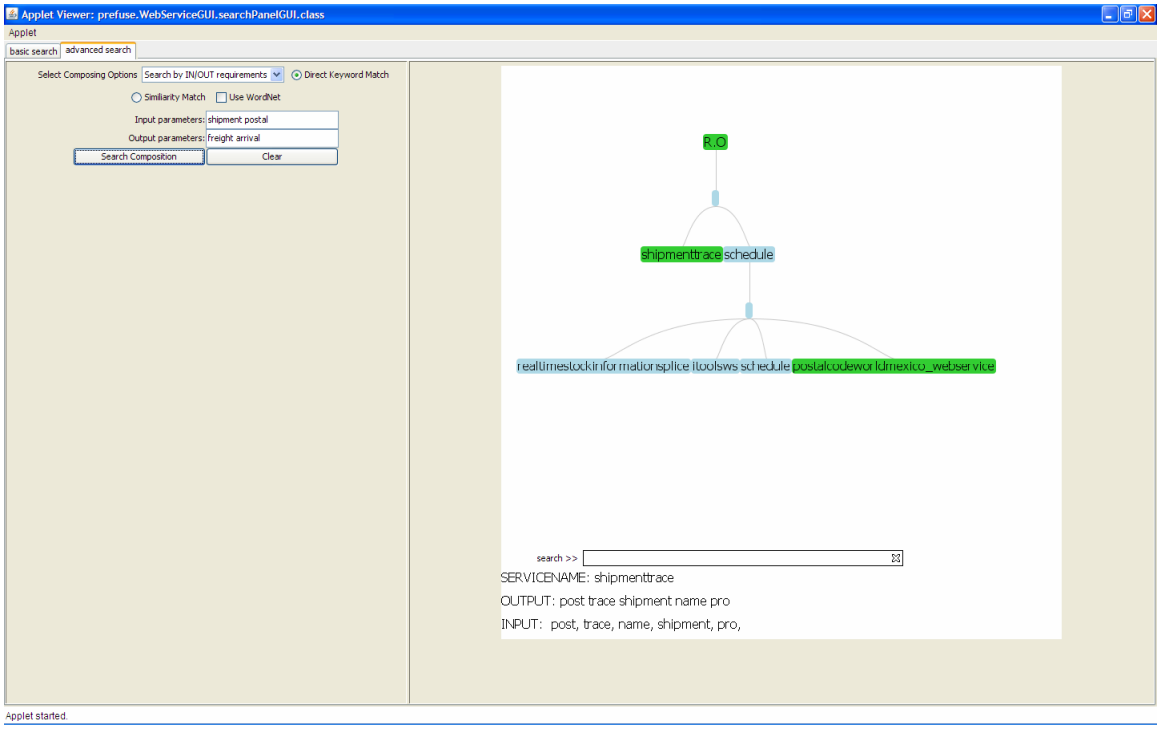

Figure 37: Composition of Input: shipment, postal Output: reight, arrival

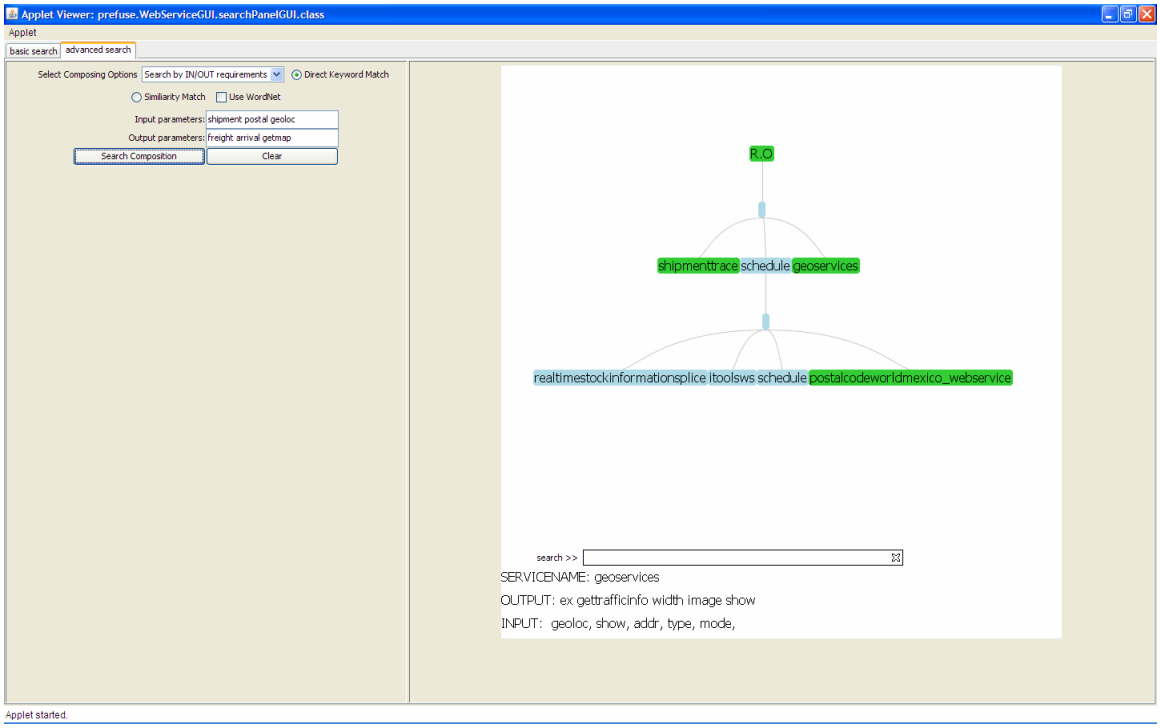

Figure 38: Composition of Input: shipment, postal, geoloc Output: freight, arrival, getmap

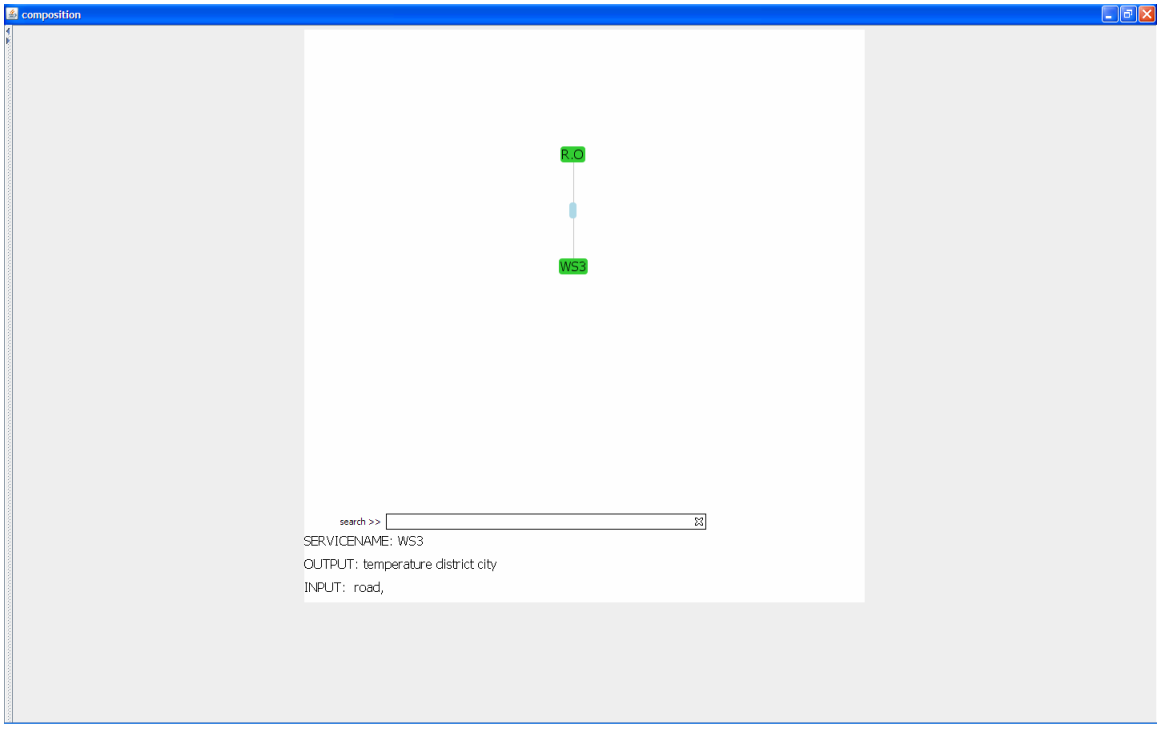

Figure 39: Composition using wordNet Input: Road, Output: territory

7. Future work

There are several other ways to improve the framework not discussed in this paper which would further enhance the capabilities of the system. Future work can extend the framework to include several extensions. One extension is to provide a publishing registry for service providers to publish their web service directly into the site and also include a tool for invoking web services directly from the framework for users to get direct access to the service and to promote more efficiently service selection.

Also, the framework can be changed to handle large user access. This can be done by moving all data calculation to the server instead of loading the framework on the client side. All preloaded data and calculation would be handled by a central server, and

clients would access the data through web services. This would allow for faster data access and searches since multiple searches done by users can be cached and reused and less duplicate calculation would be needed to be recomputed.

The crawler can also be modified to crawl addition information from a service provider's web page about the web service than just the WSDL file. This would give additional meta-data and more exact knowledge about the web service operation and functionality.

Data can be further clustered into global and local clustering of retrieved results on the client side. This can be applied if the search results get too large, clustering the result set can further put results into groups that can be easily organized for users to view.

Due to the naming convention of different service provider in naming their services, it may be possible to provide a central data bank for naming which would allow services to be better matched if the intentional parameter were the same and naming was the conflict. Composition can be further extended to find the best composition path from all possible paths and possibly calculate the cost of the composition.

8. Conclusion

As the use of web services continues to grow, the problem of searching for relevant services will be more difficult. The proposed latent semantic web service discovery and composition framework targets such a problem to provide similarity search primitives and composition of complex services enhancing the users' needs. This paper

has introduced the idea of using a novel technique of LSA combined with wordNet to extend the search based on semantic similarity in the domain of web services. The framework shows that providing a graphical user interface for users to search for web services is beneficial. Providing a specific interface to both service discovery and service composition in a single portal can allow users to find public web services easily and efficiently than current search engines. The web service discovery and composition framework explores the underlying structure of the web services operations. The framework extends from the general keyword search in that it returns web services that are relevant to the user's requirements, but do not necessarily contain the specific words that the user uses. Such examples include search of zipcode can return results of zip, zipcode, or postcode and forecast can return weather, temperature, or humidity. The implemented framework using the LSA technique on web services has shown to significantly improve the precision and recall compare to the native method. The composition search GUI also shows that new services can be composed of existing services when a service is not available.

References:

[1] Internet Growth Statistics- Today's Road to eCommerce and Global Trade, Internet World Stats. Usage and Population Statistics. Miniwatts Marketing Group. 2001-2008. http://www.internetworldstats.com/emarketing.htm

[2] Zdravko Markov and Daniel T. Larose. Data Mining the Web: Uncovering Patterns in Web Content, Structure and Usage, New Jersey: John Wiley & Sons, 2007.

[3] Web Services Activity. 2002-2007 W3C® (MIT, ERCIM, Keio), http://www.w3.org/2002/ws/

[4] Judith Hurwitz, Robin Bloor, Carol Baroudi, Marcia Kaufman. Service Oriented Architecture For Dummies. Wiley Publishing Inc, 2007.

[5] Marwan, Sabbouh, Stu Jolly, Dock Allen, Paul Silvey, Paul Denning. World Wide Web Consortium. Workshop on Web Services. April 11-12, 2001. San Jose. http://www.w3.org/2001/03/WSWS-popa/paper08

[6] Microsoft, IBM, SAP To Discontinue UDDI Web Services Registry Effort. UDDI Business Registry (UBR) Will Be Discontinued From January 12, 2006. SAP News Desk Dec.18, 2005. SYS_CON Media. SOA World Magazine. http://soa.sys-con.com/node/164624

[7] Eyhab Al-Masri and Qusay H. Mahmoud. Investigating Web Services on the World Wide Web. Refereed Track: Web Engineering – Web Service Deployment, Guelph, ON Canada. WWW 2008

[8] M.P. Singh, The Service Web. IEEE Internet Computing,vol. 4, no. 4, pp. 4–5. 2000.

[9] Tracy Gardner. An Introduction to Web Services, Oct. 02, 2001. Ariadne Issue 29 http://www.ariadne.ac.uk/issue29/gardner/intro.html

[10] Erin Cavanaugh. Web services: Benefits, challenges, and a unique, visual development solution. White Paper: Altova, Inc.. 2006

[11] Shohoud,Yasser. Real World XML Web Services. Boston: Pearson Education, Inc 2003.

[12] Bergeron, François and Louis Raymond. The advantages of electronic data interchange. ACM SIGMIS Database. pp.19-31. http://doi.acm.org/10.1145/146553.146556.

[13] Clara Yu, John Cuadrado, Maciej Ceglowski, J. Scott Payne. Patterns in Unstructured Data Discovery, Aggregation, and VisualizationA Presentation to the Andrew W. Mellon Foundation. National Institute for Technology and Liberal Education (NITLE) http://knowledgesearch.org/lsi/lsa_explanation.htm

[14] FIPS PUB 161-2 Supersedes FIPS PUB 161-1.1993 April 19

[15] A.Zhou, S.Huang, X.Wang. BITS: A Binary Tree Based Web Service Composition System. International Journal of Web Services Research. Pp. 40-58 Jan-Mar2007

[16] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. Focused crawling using context graphs. In Proceedings of 26th International Conference on Very Large Databases (VLDB), pp. 527-534, Cairo, Egypt. 2000.

[17] G. Salton, A. Wong, and C. S. Yang. A Vector Space Model for Automatic Indexing, Communications of the ACM, vol. 18, nr. 11, pp. 613–620. 1975

[18] E. Garcia. SVD and LSI Computing the Full SVD of a Matrix. http://www.miislita.com/

[19] Michael W. Berry, Zlatko Drmac, Elizabeth R.Jessup. Matrices, Vector Spaces, and Information Retrieval. Society for Industrial and Applied Mathematics. Siam Review. Vol.41, No.2, pp.335-362. 1999.

[20] Baeza-Yates, R.; Ribeiro-Neto, B. Modern Information Retrieval. New York: ACM Press, Addison-Wesley. Seiten 75 ff. 1999.

[21] Sheng Huang, Xiaoling Wang, Aoying Zhou. IEEE Efficient Web Composition Based on Syntactical Matching. 2005

[22] Marco Aiello, Chritian Platzer, Florian Rosenberg, Huy Tran, Martin Vasko, Schaharm Dustdar. Web Service Indexing for Efficient Retrieval and Composition: Distributed Systems Group, TV Vienna. 2006

[23] Seong-Chan Oh, Jung-Woon Yoo, Hyunyoung Kil, Dongwon Lee, Soundar R.T.Kumara. Semantic Web-Service Discovery and Composition Using Flexible Parameter Matching; IEEE 2007

[24] E.Al-Marsri, Q.Mahmoud . Discovering the Best Web Service, ACM 2007

[25] Zhou Zhu, James Biley, Fast Discovery of Interesting Collections of Web Services -- University of Melbourne

[26] Paolucci, M., Kawamura, T., Payne, T.R. and Sycara, K. Semantic Matching of Web Services Capabilities.Proc. International Semantic Web Conference (ISWC02), Sardinia Italy. 2002.

[27] I. Cho, J. McGregor, and L. Krause. A protocol-based approach to specifying interoperability between objects. In Proceedings of the 26th Technology of Object-Oriented Languages and Systems (TOOLS'26), pp. 3–07, Santa Barbara,CA, pp.84-96. IEEE Press. August 1998

[28] J. Purtilo and J.M. Atlee. Module Reuse by Interface Adaptation. Software Practice and Experience, Vol. 21, No. 6, pp. 539-556. 1991.

[29] The DARPA Agent Markup Language Homepage. http://www.daml.org/

[30] A. M. Zaremski and J. M. Wing. Signature Matching: a Tool for Using Softwar Libraries. ACM Transactions on Software Engineering and Methodology, Vol. 4 No. 2, pp. 146-170, Apr. 1995.

[31] A. M. Zaremski and J. M. Wing. Specifications Matching of Software Components. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, pp. 333-369, Oct. 1997.

[32] Z.Maamar, D.Benslimane, What can Context Do For Web Services? N.Narendra Communications of the ACM Dev. Vol49. No.12. 2006

[33] P.Inverardi, M.Tivoli A Reuse-based Approach to the Correct and Automatic Composition of Web-Services.; ACM Sept 4, 2007

[34] WordNet a lexical database for English language. WordNet. http://wordnet.princeton.edu/